A Hidden Herbrand Theorem: Combining the Object and Logic Paradigms

Joseph Goguen
Dept. Computer Science & Engineering
University of California, San Diego

Grant Malcolm
Department of Computer Science
University of Liverpool

Tom Kemp
Oxford

Abstract: The benefits of the object, logic (or relational), functional, and constraint paradigms can be obtained from our previous combination of the object and functional paradigms in hidden algebra, by combining it with existential queries over the states and attributes of objects, and then lifting to hidden Horn clause logic with equality, using an extension of a result due to Diaconescu. We call this novel programming paradigm *active constraint object programming*, suggest some applications for it, and show that it is computationally feasible by reducing it to familiar problems over term algebras (i.e., Herbrand universes). Our main result is a version of Herbrand's Theorem, lifted from hidden algebra by the extended result of Diaconescu. This paper also contains new results on the existence of initial and final models, and on the consistency of hidden theories.

1 Introduction

The object paradigm has many practical advantages (e.g., see [51]), including its support for code reuse through inheritance, its intuitive appeal, and its affinity for data abstraction and concurrency. However, it has not been integrated with the complementary advantages of the logic (or more accurately, relational), functional, and constraint paradigms. The advantages of these paradigms include clean declarative semantics, and (for the relational case) natural integration with database query languages. Logic programming and functional programming over user-definable abstract data types were combined in [34] by combining their underlying logics (Horn clause logic and equational logic, respectively), to obtain Horn clause logic with equality, which in addition provides an elegant semantics for contraint logic programming. Following the suggestion in [34] that the best way to combine paradigms is to *combine their underlying logics*, we here extend the combination of relational and functional paradigms, by extending Horn clause logic with equality (as in [34, 35]) to *hidden Horn clause logic* with equality, building on prior work on *hidden algebra* as a foundation for the object paradigm [19, 24, 31]. Hidden algebra is a natural extension of the initial algebra approach to abstract data types (ADTs) [39] that handles states in a more natural way, and also supports behavioral correctness and refinement proofs for systems with objects, inheritance, nondeterminism and concurrency, in addition to functional programming [15, 38, 57, 58, 25]. The main result of this paper is a Herbrand theorem for hidden Horn clause logic, allowing solutions of queries to be constructed in a term algebra; it is obtained by applying the extended lifting result of Diaconescu [10] to the Hidden equational case.

All this provides a semantic foundation for a novel programming paradigm, in which posing a query can activate methods that change the world so that a solution actually comes to exist [36, 20, 27]. For example, consider a query about a holiday package, where a customer has provided constraints on destination, cost, flight times, seat assignments, hotels, expected weather, etc.; a solution to this query would be an actual travel package, with actual tickets, reservations, visas, etc., satisfying the constraints (of course, subject to customer approval before commitment). Another example might be a query about an operating system, with constraints on processor speed, hard drive size, core memory, external connections, etc.; the solution would then be an actual properly configured operating system, assembled using generic components from a software library, satisfing all the constraints and ready to run. Other examples come from flexible manufacturing, where queries could create customized cars, rugs, furniture, and even clothing. One can easily imagine

many other applications, where answering queries over some domain-specific objectbase actually creates new objects that satisfy the given constraints. We call this new paradigm *active constraint object programming*; it seems particularly well suited to new applications arising in connection with the world wide web and mobile computing.

Whereas classical initial algebra semantics for ADTs [39] requires modelling states in an explicit functional (i.e., visible) style, hidden algebra allows states to be left implicit (i.e., hidden), as in most real programming, including of course imperative languages, but also object oriented languages, for which it is a particularly good match. Hidden algebra differs from classical algebraic approaches in declaring some sorts to be *hidden*, for modeling *states*, while others model *data* in the classical way [39]; states cannot be observed directly, but only indirectly through the visible results of experiments, which consist of applying a sequence of methods and then examining an attribute. Hidden algebra originated in [19], extending earlier work of Goguen and Meseguer on (what they then called) *abstract machines* [33, 50], mainly through using *behavioral satisfaction* for equations, an idea introduced by Reichel [54] in the context of partial algebras. Reichel [55] later introduced behavioral equivalence for states, which is also used in hidden algebra.

Goguen [19] showed that hidden algebra with some intuitive restrictions on signature maps forms an institution (in the sense of [22], i.e., a logical system) and used this to model objects; combined with results in this paper, this implies that parameterized programming [17, 18, 47], with its powerful generic module facilities, is available for our new paradigm. Two restrictions are that operations should have at most one argument of hidden sort, and that a fixed visible sorted algebra should be part of every model (as discussed in Section 3); however, later work has relaxed these restrictions, while still yielding an institution [15, 58, 38, 57, 25]. Algebraic approaches have the advantage of relatively easy reasoning, because of the many well developed computational techniques available, e.g., term rewriting, unification, and narrowing.

Section 2 is a condensed review of overloaded many sorted algebra, mainly to fix notation and terminology, but also to introduce some lemmas about substitution. Section 3 introduces hidden algebra, including the behavioral satisfaction of conditional equations. This section also characterizes behavioral equivalence in a way that serves as a basis for *coinduction*, a powerful behavioral proof technique, as in work of Goguen and Malcolm [26, 48, 31] and Goguen and Roşu [58, 38, 57, 25]. Section 3 also contains new results on the consistency of hidden theories. Section 4 presents basic results for reasoning about hidden algebraic specifications, showing that ordinary equational deduction is sound, that behavioral satisfaction of an equation reduces to satisfaction by an initial algebra for certain classes of reachable models, and that the restriction to reachable models is not required for ground equations.

Section 5 treats existential behavioral equational queries, which have the form $(\exists X)\, t_1 = t'_1, \ldots, t_m = t'_m$ where the equalities serve as constraints on the possible values for logical variables in the set X. A solution to such a query in an algebra A consists of values in A for each variable (some of which may range over states) such that each equation is behaviorally satisfied. The classical Herbrand Theorem [41] says that for the models of a set of Horn clauses, existential queries can be answered by examining a particular term model, called the Herbrand universe (see [44, 1] for overviews of logic programming). This result was generalized to Horn clause logic with equality by Goguen and Meseguer [34, 35], showing that it suffices to examine a term model, and moreover, that this model is initial. The advantage of a term model is the well-established techniques for equational computation that are available for it. Our hidden Herbrand Theorem states that if a query is behaviorally satisfied by a certain term algebra, then it is behaviorally satisfied by all hidden algebras, which means any correct implementation of the underlying database and functionality. Section 6 generalizes a result of Diaconescu [10] which allows us to lift the result from hidden algebra to hidden Horn clause logic with equality.

Acknowledgements

The research reported in this paper was largely conducted between 1994 and 1996, with partial support from the UK Science and Engineering Research Council, the European Union under ESPRIT-2 BRA Working Groups 6071, IS-CORE (Information Systems COrrectness and REusability) and 6112, COMPASS (COMPrehensive Algebraic Approach to System Specification and development), Fujitsu Laboratories Limited, and a contract under management of the Information Technology Promotion Agency (IPA) of Japan, as part

of the Industrial Science and Technology Frontier Program 'New Models for Software Architectures.' We thank members of the Declarative Group at Oxford University, particularly Răzvan Diaconescu, for valuable comments, and Frances Page for work on the original file. The paper was edited at UCSD mostly in May and June 2000, with partial support from National Science Foundation grant CCR-9901002; the main non-cosmetic change is a significant improvement by Grigore Roşu to results on consistency in Section 3.3.1; we greatly thank him for this, as well as for other comments, which have helped to improve the paper.

2 Prerequisites, Notation and Preliminaries

We assume familiarity with many sorted algebra, but to establish notation, we will briefly review some main concepts and results. For compatible expositions with more detail, see [28] or [50]; this approach, based on indexed sets, originated in lectures by Joseph Goguen at the University of Chicago in 1968. Some of the examples in Sections 4.2 and 5 assume basic knowledge of term rewriting, such as confluence and termination. Introductions to term rewriting may be found in [9, 21], among other places. Occasionally it is convenient to express a result or construction in the vocabulary of category theory, but we use only very basic notions like category, functor, and initial object. Readers unfamiliar with these need not worry, because none of our constructions or proofs employ any category theory. We use boldface to denote categories, e.g., $\mathbf{C}$. Given morphisms $f\colon A \to B$ and $g\colon B \to C$, we let $g \circ f$ denote their composition, a morphism $A \to C$, and we let 1_A denote the identity morphism at an object A. See [2, 43] for introductions to category theory.

An S-**indexed** (or **sorted**) **set** A is a family $\{A_s \mid s \in S\}$ of sets indexed by the elements of S. An S-**indexed** (or **sorted**) **function** $f\colon A \to B$ is a family $\{f_s\colon A_s \to B_s \mid s \in S\}$; similarly, an S-sorted relation $R \subseteq A \times B$ is a family $\{R_s \subseteq A_s \times B_s \mid s \in S\}$. All operations on sets extend to operations on S-sorted sets, for example $A \subseteq B$ means that $A_s \subseteq B_s$ for each $s \in S$.

Given a set S, we let S^* denote the set of all finite sequences of elements from S, and we let $[]$ denote the empty sequence of elements from S. Given an S-indexed set A and $w = s_1 \ldots s_n \in S^*$, let $A_w = A_{s_1} \times \cdots \times A_{s_n}$; in particular, let $A_{[]} = \{\star\}$, some singleton set.

A **signature** (S, Σ) is an $S^* \times S$-indexed set $\Sigma = \{\Sigma_{w,s} \mid w \in S^*,\ s \in S\}$; we often write just Σ instead of (S, Σ). Notice that this definition permits *overloading*, in that the sets $\Sigma_{w,s}$ need *not* be disjoint; this can be useful for application to dynamic binding in the object paradigm. A **signature morphism** φ from a signature (S, Σ) to a signature (S', Σ') is a pair (f, g) consisting of a map $f\colon S \to S'$ of sorts and an $S^* \times S$-indexed family of maps $g_{w,s}\colon \Sigma_{w,s} \to \Sigma'_{f^*(w),f(s)}$ on operation symbols, where $f^*\colon S^* \to S'^*$ is the extension of f to strings defined by $f^*([]) = []$ and $f^*(ws) = f^*(w)f(s)$, for w in S^* and s in S. We write $\varphi(s)$ for $f(s)$, $\varphi(w)$ for $f^*(w)$, and $\varphi(\sigma)$ for $g_{w,s}(\sigma)$ when $\sigma \in \Sigma_{w,s}$.

A Σ-**algebra** A consists of an S-indexed set A and a function $A_\sigma\colon A_w \to A_s$ for each $\sigma \in \Sigma_{w,s}$; the set A_s is called the **carrier** of A of sort s. A Σ-**homomorphism** from a Σ-algebra A to another B is an S-indexed function $f\colon A \to B$ such that

$$f_s(A_\sigma(a_1, ..., a_n)) = B_\sigma(f_{s_1}(a_1), ..., f_{s_n}(a_n))$$

for each $\sigma \in \Sigma_{w,s}$ with $w = s_1 \ldots s_n$ and $a_i \in A_{s_i}$ for $i = 1, \ldots, n$. (When $n = 0$, i.e., when $w = []$, the condition is simply that $f(A_\sigma) = B_\sigma$.) Let $\mathbf{Alg}_\Sigma$ denote the category with Σ-algebras as objects and Σ-homomorphisms as morphisms.

Given a subsignature $\Psi \subseteq \Sigma$, there is a **reduct** functor, $\restriction_\Psi\colon \mathbf{Alg}_\Sigma \to \mathbf{Alg}_\Psi$, traditionally written using postfix notation, that sends a Σ-algebra A to $A\restriction_\Psi$, which is A viewed as a Ψ-algebra by forgetting about any sorts and operations in Σ that are not in Ψ; similarly, if $f\colon A \to B$ is a Σ-homomorphism, then $f\restriction_\Psi\colon A\restriction_\Psi \to B\restriction_\Psi$ is the Ψ-homomorphism obtained by restricting f to the sorts in Ψ.

Given a many sorted signature Σ and an S-indexed set (of variable symbols) X (where the sets X_s are disjoint), we let $T_\Sigma(X)$ denote the (S-indexed) **term algebra** with operation symbols from Σ and variable symbols from X; it is the **free** Σ-algebra generated by X, in the sense that if $\theta\colon X \to A$ is an **assignment**, i.e., a (many sorted) function to a Σ-algebra A, then there is a unique extension of θ to a Σ-homomorphism $\theta^*\colon T_\Sigma(X) \to A$. (Strictly speaking, the usual term algebra is not free unless each term has a unique parse; however, even if this is not the case, the closely related term algebra with constants annotated by their

sort, is free.) Also, we let T_Σ denote the **initial** term Σ-algebra, $T_\Sigma(\emptyset)$, recalling that there is a unique Σ-homomorphism $!_A : T_\Sigma \to A$ for any Σ-algebra A (this homomorphism simply interprets any ground term as an element of A). Call $t \in T_\Sigma$ a **ground Σ-term**. Given a ground Σ-term t, let t_A denote the element $!_A(t)$ in A. Call A **reachable** iff $!_A$ is surjective, i.e., iff each element of A is 'named' by at least one ground term.

A special case of free extension is **substitution**, where the target algebra A is a term algebra, often $T_\Sigma(X)$ itself. We will often use the following property of free extensions:

Lemma 1: Given an assignment $\theta : X \to A$ and a Σ-homomorphism $f : A \to B$, then

$$(f \circ \theta)^* = f \circ \theta^* : T_\Sigma(X) \to B \ .$$

Proof: By freeness, there is only one Σ-homomorphism from $T_\Sigma(X)$ to B that extends $f \circ \theta$. □

A **conditional Σ-equation** consists of a variable set X, terms $t, t' \in T_\Sigma(X)_s$ for some sort s, and terms $t_j, t'_j \in T_\Sigma(X)_{s_j}$ for $j = 1, ..., m$, by convention written in the form

$$(\forall X)\ t = t' \ \textbf{ if } \ t_1 = t'_1, ..., t_m = t'_m \ .$$

The special case where $m = 0$ is called an (**unconditional**) **equation**, written $(\forall X)\, t = t'$. A **ground equation** has $X = \emptyset$.

A Σ-algebra A **satisfies** a conditional equation, written

$$A \models_\Sigma (\forall X)\, t = t' \ \textbf{ if } \ t_1 = t'_1, ..., t_m = t'_m \ ,$$

iff for all $\theta : X \to A$, we have $\theta^*(t) = \theta^*(t')$ whenever $\theta^*(t_j) = \theta^*(t'_j)$ for $j = 1, ..., m$. Given a set E of (possibly conditional) Σ-equations, let $\mathbf{Alg}_{\Sigma,E}$ denote the full subcategory of $\mathbf{Alg}_\Sigma$ with objects the Σ-algebras that satisfy E; we call these (Σ, E)-algebras.

A **Σ-congruence** on a Σ-algebra A is an S-sorted family of relations, $\equiv_s$ on A_s, each of which is an equivalence relation and that also satisfy the **congruence property**, that given $\sigma \in \Sigma_{s_1 ... s_n, s}$, and given $a_i, a'_i \in A_{s_i}$ for $i = 1, ..., n$, then

$$A_\sigma(a_1, ..., a_n) \equiv_s A_\sigma(a'_1, ..., a'_n) \quad \text{whenever} \quad a_{s_i} \equiv_{s_i} a'_{s_i} \text{ for } i = 1, ..., n \ .$$

The **quotient** of A by a Σ-congruence $\equiv$, denoted $A/_\equiv$, has

$$(A/_\equiv)_s = A_s/_{\equiv_s}$$

and inherits a Σ-algebra structure by defining

$$(A/_\equiv)_\sigma([a_1], \ldots, [a_n]) = [A_\sigma(a_1, \ldots, a_n)] \ ,$$

where $\sigma \in \Sigma_{s_1 ... s_n, s}$ and $a_i \in A_{s_i}$, and where $[a]$ denotes the $\equiv$-equivalence class of a. We can define a **quotienting** homomorphism $q : A \to A/_\equiv$ by $q_s(a) = [a]$ for each element $a \in A_s$ for $s \in S$, which is clearly surjective.

Lemma 2: Let A be a Σ-algebra with $\equiv$ a Σ-congruence on A. For $(\forall X)\, t = t'$ an unconditional Σ-equation, we have

$$A \models_\Sigma (\forall X)\, t = t' \ \text{ implies } \ A/_\equiv \models_\Sigma (\forall X)\, t = t' \ .$$

Proof: Let q be the quotienting homomorphism $A \to A/_\equiv$. For any assignment $\theta : X \to A/_\equiv$, because q is surjective, for every $s \in S$ and each $x \in X_s$, we can choose an element $a_x \in A_s$ such that $q_s(a_x) = \theta(x)$. This defines an assignment $\theta_q : X \to A$ by $\theta_q(x) = a_x$, with the property that $q \circ \theta_q = \theta$.

If $A \models_\Sigma (\forall X)\, t = t'$, then using Lemma 1, we have

$$\theta^*(t) = (q \circ \theta_q)^*(t) = q(\theta_q^*(t)) = q(\theta_q^*(t')) = (q \circ \theta_q)^*(t') = \theta^*(t') \ .$$

Because θ is arbitrary, this shows $A/_\equiv \models_\Sigma (\forall X)\, t = t'$ as desired. □

We now consider the *logic* of many sorted algebra, that is, rules that can be used to deduce new equations from old equations.

Definition 3: Given a set E of (possibly conditional) Σ-equations, we define the (unconditional) Σ-equations that are **derivable** from E recursively, by the following **rules of deduction** for many sorted equational logic:

(0) Base: Each unconditional equation in E is derivable.

(1) Reflexivity: Each equation $(\forall X)\, t = t$ is derivable.

(2) Symmetry: If $(\forall X)\, t = t'$ is derivable, then so is $(\forall X)\, t' = t$.

(3) Transitivity: If $(\forall X)\, t = t'$ and $(\forall X)\, t' = t''$ are derivable, then so is $(\forall X)\, t = t''$.

(4) Congruence: If $(\forall X)\, t_i = t'_i$ is derivable, where $t_i, t'_i \in T_\Sigma(X)_{s_i}$ for $i = 1, ..., n$, then for any $\sigma \in \Sigma_{s_1 \ldots s_n, s}$, the equation $(\forall X)\, \sigma(t_1, \ldots, t_n) = \sigma(t'_1, \ldots, t'_n)$ is also derivable.

(5) Modus Ponens: Given $(\forall X)\, t = t'$ **if** $t_1 = t'_1, \ldots, t_m = t'_m$ in E and given a substitution $\theta\colon X \to T_\Sigma(Y)$ such that $(\forall Y)\, \theta^*(t_j) = \theta^*(t'_j)$ is derivable for $j = 1, ..., m$, then $(\forall Y)\, \theta^*(t) = \theta^*(t')$ is also derivable.

Given a set E of Σ-equations, let $E^\Box$ denote the set of all unconditional Σ-equations derivable from E using the above rules, and call it the **deductive closure** of E. Also, let $E^{\bigcirc}$ be the S-sorted set of pairs (t, t') of *ground* Σ-terms such that $(\forall \emptyset)\, t = t'$ is derivable from E. Note that $E^{\bigcirc}$ is a Σ-congruence by rules (1)–(4). $\Box$

The following completeness result for conditional many-sorted algebra was first proved by Goguen and Meseguer [33], though the unconditional one-sorted case is well known, going back to Birkhoff [3] in 1935:

Theorem 4: Given a set E of (possibly conditional) Σ-equations, an unconditional Σ-equation is satisfied by every (Σ, E)-algebra iff it is derivable from E using the rules (0)–(5). $\Box$

We use the standard notation $E \models_\Sigma e$ to indicate that all (Σ, E)-algebras satisfy the equation e. Goguen and Meseguer [33] use the theorem above to prove the following basic result:

Theorem 5: The quotient algebra $T_\Sigma/_{E^{\bigcirc}}$ is initial in $\mathbf{Alg}_{\Sigma,E}$. $\Box$

Of course, there are many other initial (Σ, E)-algebras, but they are all Σ-isomorphic to this one; this fact expresses the *abstractness* of abstract data types as initial (Σ, E)-algebras [39]. Mathematical expositions usually emphasize the more general existence of free algebras, but this follows easily from Theorem 5. Many interesting results about conditional equations and their algebras appear in the literature for the one sorted case, e.g. [7, 45, 59, 49], but as far as we know, few of these have been considered carefully for the many sorted case.

3 Hidden Algebra

Well-designed software often fails to strictly satisfy its specifications, but instead satisfies them only *behaviorally*, in the sense that they *appear* to be true under all possible experiments. Hidden algebra extends prior work on abstract data types and algebraic specification to behavioral satisfaction in a surprisingly simple way that also handles internal states, objects, inheritance, concurrency, nondeterminism, and more. The most important results in this theory are powerful coinduction proof rules, which support behavioral correctness and refinement proofs that are considerably simpler than proofs done with more classical methods. For more details, see [15, 38, 57, 58, 25].

Hidden algebra captures the fundamental distinction between basic data types used as values for attributes (integers, Booleans, characters, etc.) and internal states by modeling the former with *visible* sorts and the

latter with *hidden* sorts. The various components of a complex system must share the same representations for basic data, or else they will not be able to communicate; therefore it makes sense to work with a fixed collection of data values, which can be bundled together to form a fixed algebra. Our assumptions and notation for these data values are given in the following:

Definition 6: Let D be a fixed algebra, let Ψ be its signature, let V be its sort set, and assume that $D_v \subseteq \Psi_{[],v}$ for each $v \in V$. We may call (V, Ψ, D) the **visible data universe** and D the **data algebra**. □

The assumption $D_v \subseteq \Psi_{[],v}$ just means that we have a distinct name for each ground data value; this is reasonable, and is needed in our construction of a final algebra. It is possible to strengthen the assumption to $D_v = \Psi_{[],v}$, but the added generality of Definition 6 does no harm. The fixed data universe (V, Ψ, D) is thought of as containing the basic data types where V contains the names of their sorts, Ψ contains the (names of) operations for these sorts, and D provides an appropriate (it need not necessarily be initial) interpretation for these sorts and operations. The examples in this paper assume a data algebra containing at least the natural numbers and Booleans. Signatures for hidden algebra are defined with respect to a given visible data universe:

Definition 7: A **hidden signature** (over (V, Ψ, D)) is a pair (H, Σ) such that $(V \cup H, \Sigma)$ is a many sorted signature with $\Psi \subseteq \Sigma$ and $H \cap V = \emptyset$, and such that the following two conditions hold:

(S1) if $w \in V^*$ and $v \in V$, then $\Sigma_{w,v} = \Psi_{w,v}$;

(S2) for each $\sigma \in \Sigma_{w,s}$, at most one element of w is in H.

We often abbreviate (H, Σ) to Σ, and write S for $V \cup H$.

The elements of V are referred to as **visible sorts**, and elements of H as **hidden sorts**. If $w \in S^*$ contains a hidden sort, then $\sigma \in \Sigma_{w,s}$ is called a **method** if $s \in H$, and an **attribute** if $s \in V$. If $w \in V^*$ and $s \in H$, then $\sigma \in \Sigma_{w,s}$ is called a (**generalized**) **hidden constant**. □

Condition (S1) expresses *data encapsulation*, in the sense that if $\Psi \subseteq \Sigma$ is a module inclusion, then new operations on old data are disallowed. Condition (S2) says that methods and attributes act on single states, corresponding to the natural locality of states in object oriented programming; it is needed for some results in Section 3.1, as well as for final models to exist [5]; however, many other results of hidden algebra generalize to multiple hidden arguments [15, 58, 38, 57, 25].

Definition 8: Given hidden signatures Σ and Σ', a **hidden signature morphism** $\Phi : \Sigma \to \Sigma'$ is a signature morphism $\Phi = (f, g) : \Sigma \to \Sigma'$ such that:

(M1) $f(v) = v$ for $v \in V$;

(M2) $f(H) \subseteq H'$ (where H' is the hidden sort set of Σ');

(M3) $g(\psi) = \psi$ for $\psi \in \Psi$; and

(M4) if $\sigma' \in \Sigma'_{w',s'}$ and some sort in w' lies in $f(H)$, then $\sigma' = g(\sigma)$ for some $\sigma \in \Sigma$.

□

The first three conditions say that hidden signature morphisms preserve both visibility and hiddenness for both sorts and operations, while the fourth expresses the encapsulation of classes, in the sense that no new methods or attributes can be defined on an imported class. It is not difficult to check that hidden signatures and their morphisms form a category [19].

Definition 9: A **hidden Σ-algebra** is a Σ-algebra A such that $A|_\Psi = D$. A **hidden Σ-homomorphism** $f : A \to A'$ is a Σ-homomorphism such that $f|_\Psi = 1_D$. Let $\mathbf{HAlg}_\Sigma$ denote the category of all hidden Σ-algebras and their hidden Σ-homomorphisms. □

Hidden satisfaction can be quite different from satisfaction in many sorted algebra; intuitively, an equation is satisfied in hidden algebra if its left and right sides are indistinguishable by any experiment which produces a visible sorted result. This notion of experiment is made precise by the following notion of *context*:

Definition 10: Given a hidden signature (H, Σ) and a sort $s \in S$, a Σ-**ground context** of sort s is a visible sorted Σ-term having a *single occurrence* of a new variable symbol z of sort s. A **general context** is a term in $T_\Sigma(X \cup \{z\})_v$ for some set X of variable symbols and v a visible sort. For simplicity, we often use just the word **context** to refer to a ground context. A general context is **appropriate** for a term t iff the sort of t matches the sort of z; we write $c[t]$ for the result of substituting t for z in the context c. We let $T_\Sigma[z]$ denote the V-indexed set of contexts using the variable z, and sometimes we may write a context c as $c[z_s]$ to indicate that the sort of the variable z is s. □

Definition 11: A hidden Σ-algebra A **behaviorally satisfies** a Σ-equation $(\forall X)\, t = t'$ iff for all appropriate ground contexts $c \in T_\Sigma[z]$, we have

$$A \models_\Sigma (\forall X)\, c[t] = c[t'] \ .$$

In this case we write $A \models\!\equiv_\Sigma (\forall X)\, t = t'$, and we often omit Σ from $\models\!\equiv_\Sigma$. Note that this is equivalent to

$$\theta^*(c[t]) = \theta^*(c[t'])$$

for all interpretations $\theta : X \to A$ and all appropriate ground contexts c.

Similarly, A **behaviorally satisfies** a conditional equation e of the form

$$(\forall X)\, t = t' \ \textbf{if} \ \ t_1 = t'_1, ..., t_m = t'_m \ \ ,$$

also written $A \models\!\equiv_\Sigma e$, iff for every interpretation $\theta : X \to A$, we have

$$\theta^*(c[t]) = \theta^*(c[t'])$$

for all appropriate ground contexts c whenever

$$\theta^*(c_j[t_j]) = \theta^*(c_j[t'_j])$$

for $j = 1, ..., m$, and for all appropriate ground contexts c_j.

A **hidden** (or **behavioral**) **theory** (or **specification**) is a triple (H, Σ, E), where (H, Σ) is a hidden signature and E is a set of Σ-equations; we may write (Σ, E) for short. A **model** of a hidden theory $P = (H, \Sigma, E)$ is a hidden Σ-algebra A that behaviorally satisfies each equation in E. Such a model is also called a **hidden** (Σ, E)**-algebra**, or a hidden P-algebra, and in this case we may write $A \models\!\equiv P$ or $A \models\!\equiv_\Sigma E$. Given any set E of Σ-equations, $\mathbf{HAlg}_{\Sigma,E}$ denotes the full subcategory of $\mathbf{HAlg}_\Sigma$ with objects hidden (Σ, E)-algebras. Let $E \models\!\equiv_\Sigma e$ mean that $A \models\!\equiv_\Sigma E$ implies $A \models\!\equiv_\Sigma e$ for all hidden Σ-algebras A, where e is a Σ-equation. We will call the relation $\models\!\equiv_\Sigma$ **behavioral satisfaction**, or sometimes **hidden satisfaction**. □

Notice that we cannot define the behavioral satisfaction of conditional equations in the same simple style used in the first definition for unconditional equations above, because the same interpretation must be used for the conclusion and each condition, whereas the contexts must be allowed to be different for each condition.

A hidden (Σ, E)-algebra A can be seen as a way of implementing objects in the class defined by the specification (Σ, E); elements of hidden sort in A are the possible states of such objects in this implementation. The collection of all (Σ, E)-algebras gives all the possible implementations for this class, and is our intended denotation for the specification (Σ, E). This is usually referred to as *loose* semantics, though more precisely, it is loose semantics over a fixed (or "protected") data universe. This semantics is used throughout this paper, but other approaches are possible, as for example in [38].

Example 12: We specify a simple flag object, where intuitively a flag is either up or down, and there are methods to put it up, to put it down, and to reverse its state. We use the notation of OBJ3 (see [40, 30], although here equality for hidden sorts has a very different meaning than in standard OBJ):

```
th FLAG is sort Flag .
   pr DATA .
   ops (up_) (dn_) (rev_) : Flag -> Flag .
   op up?_ : Flag -> Bool .
   var F : Flag .
   eq up? up F = true .
   eq up? dn F = false .
   eq up? rev F = not up? F .
endth
```

Here `FLAG` is the name of the module and `Flag` is the name of the class of flag objects. The line `pr DATA` indicates the "protecting" importation of a specification for the visible data universe, providing a signature (V, Ψ) which contains at least the sort `Bool` with operations `true` and `false`, for which we have implicitly chosen some fixed interpretation D, so that we have a fixed data universe (V, Ψ, D). The operations `up`, `dn` and `rev` are methods which change the state of flag objects, and `up?` is an attribute that tells whether or not the flag is up. The three methods and one attribute all have prefix syntax (the underscore '_' indicates where arguments are placed).

Note that each equation specifies the value of the `up?` attribute after application of some method. This idiom is especially useful in hidden algebra because it specifies the value of the attribute for any state of the form $m(\mathtt{F})$, where m is any method; an alternative approach such as specifying `rev` with the equations

```
eq  rev up F  =  dn F .
eq  rev dn F  =  up F .
```

would only determine values for `up? rev F` when `F` has the form `up F'` or `dn F'`. □

If Σ is the signature of `FLAG`, then a model of `FLAG` is a Σ-algebra whose restriction to Ψ is D, that provides functions for all the methods and attributes in Σ, and that behaves as if it satisfies the given equations. Elements of such models are possible *states* for `Flag` objects.

Example 13: The stack example is ubiquitous, and therefore provides a particularly good benchmark for comparing specification formalisms. We assume that the data universe specified in `DATA` contains at least the natural numbers with sort `Nat`.

```
th STACK is sort Stack .
  pr DATA .
  op push : Nat Stack -> Stack .
  op top_ : Stack -> Nat .
  op pop_ : Stack -> Stack .
  op empty : -> Stack .
  var S : Stack .
  var N : Nat .
  eq pop push(N,S) = S .
  eq top push(N,S) = N .
  eq pop empty = empty .
  eq top empty = 0 .
endth
```

The first line gives the name `STACK` to this specification, and declares the hidden sort `Stack`. After that, the data specification `DATA` is imported, and then the next four lines declare operation symbols for pushing values onto the stack, looking at the value on the top of the stack, popping the top value from the stack, plus a constant for the empty stack. The behaviou of these operations is described by the equations.

The algebras for `STACK` need only *appear* to satisfy its equations when observed through contexts, which necessarily have `top` as their head operation. In particular, the common implementation of a stack by a pointer and an array does *not* actually satisfy the equation

$$\mathtt{pop\ push(N,S)} = \mathtt{S}\ ,$$

but it *does* satisfy it behaviorally, and is therefore a STACK-algebra. A detailed mechanical correctness proof for this implementation of stack can be found at

http://www.cse.ucsd.edu/groups/tatami/kumo/exs/stack/

It is similar to the hand proof sketched in [29]. □

See [31, 29] for a more general and comprehensive introduction to hidden algebra; the first paper on this subject [19] showed that equations as sentences, hidden algebras as models, and behavioral satisfaction, give an institution (in the sense of [22]), and in particular, that the Satisfaction Condition holds, which intuitively means that truth is invariant under change of notation; an alternative institution is given in [38]; although the present paper makes no use of institutions, the advantage of making this observation is that all of the machinery of parameterized programming [17, 18] becomes available, including its powerful generic module mechanism based on views.

3.1 Using More Restricted Contexts

This section shows that hidden satisfaction can be defined using smaller classes of contexts, i.e., using more restricted experiments to distinguish states. In fact, Definition 11 above for hidden satisfaction already uses fewer contexts than the original definition in [19], which defined satisfaction using general contexts. We will show that both of these definitions are equivalent to definitions of behavioral satisfaction that use certain even more restricted contexts:

Definition 14: A $\Sigma(X)$-term is **local** iff every proper visible subterm is a $\Psi(X)$-term. A Σ-context is **local** iff every proper visible subterm is in D. We write $L_\Sigma(X)$ for the S-indexed set of local $\Sigma(X)$-terms, L_Σ for $L_\Sigma(\emptyset)$, and $L_\Sigma[z]$ for the V-indexed set of local Σ-contexts. An equation is **local** iff its left and right sides are local terms, and its conditions (if any) are visible sorted and use only Ψ-operations; a set of equations is local iff each one is local. □

Note that every $\Psi(X)$-term is local. The reason for placing a stronger restriction on contexts, that proper visible subterms are in D, is that that we want to make this set of local contexts as small as possible (cf. Proposition 19 below). Here are some examples over the stack signature:

`top push(top empty, empty),  1 + top empty`

are not local terms, whereas

`top push(0, empty),  top push(1 +` x`, empty)`

are local terms, and

`top push(1,` z`),  1 + top` z

are a local and non-local context, respectively. Before showing the equivalence of definitions of behavioral satisfaction for (conditional) equations using various kinds of context, we need one more definition and two lemmas.

Definition 15: Given a Σ-algebra A and an assignment $\theta : X \to A$, any general context c gives rise to a function $\theta^* c \colon A \to A$ defined by interpreting in A each operation that occurs in the context c; that is, for each element a of A, we have $(\theta^* c)(a) = \theta^*_a(c)$, where $\theta_a : X \cup \{z\} \to A$ extends θ by $\theta_a(z) = a$. When c is a ground context, we write A_c for $(!_A)^* c$, where $!_A$ is the unique S-sorted mapping $\emptyset \to A$. □

Lemma 16: Given a Σ-algebra A, a $\Sigma(X)$-term t, an appropriate general context c, and an assignment $\theta \colon X \to A$, then $\theta^*(c[t]) = (\theta^* c)(\theta^*(t))$. In particular, if c is ground, then $\theta^*(c[t]) = A_c(\theta^*(t))$. Moreover, if $A = T_\Sigma$ and c is ground, then $\theta^*(c[t]) = c[\theta^*(t)]$.

Proof: The first assertion is by induction on the structure of contexts c, the second by definition of A_c, and the third because $A_c(z) = c[z]$ in this case, since $\theta^* c = c$. □

Lemma 17: For any general context c, hidden algebra A, and assignment $\theta : X \to A$, there exists a local ground context l such that for every $a, a' \in A$, if $A_l(a) = A_l(a')$ then $(\theta^* c)(a) = (\theta^* c)(a')$.

Proof: Let $c \in T_\Sigma(X)[z]$ be a general context, and let c_0 be the smallest subterm of c that has visible sort and contains z. Clearly, if $(\theta^* c_0)(a) = (\theta^* c_0)(a')$ then $(\theta^* c)(a) = (\theta^* c)(a')$. If z is of visible sort, then $c_0 = z$ is local, and we can take l to be c_0. Otherwise, if z is of hidden sort, it follows from condition (S2) in Definition 7 that every hidden sorted variable from X that occurs in c_0 occurs within a visible sorted subterm of c_0 that does not contain z. Let l be the result of replacing every visible sorted subterm of c_0 that does not contain z with its image under θ^*; then l is a ground local context and $\theta^* c_0 = A_l$, so that if $A_l(a) = A_l(a')$ then $(\theta^* c_0)(a) = (\theta^* c_0)(a')$, and therefore $(\theta^* c)(a) = (\theta^* c)(a')$. □

Now we show that defining behavioral satisfaction with general contexts is equivalent to our earlier definitions with just ground contexts:

Proposition 18: A conditional equation $(\forall X)\ t = t'$ **if** $t_1 = t'_1, \ldots, t_m = t'_m$ is behaviorally satisfied by a hidden algebra A iff for all $\theta : X \to A$, whenever $\theta^*(c_j[t_j]) = \theta^*(c_j[t'_j])$ for $j = 1, \ldots, m$ and for all appropriate general contexts c_j, then $\theta^*(c[t]) = \theta^*(c[t'])$ for all appropriate general contexts c.

Proof: Suppose A behaviorally satisfies the conditional equation above, and let $\theta : X \to A$ be such that $\theta^*(c_j[t_j]) = \theta^*(c_j[t'_j])$ for $j = 1, \ldots, m$ and all appropriate general contexts c_j. Then because every ground context is a general context, we have $\theta^*(c_j[t_j]) = \theta^*(c_j[t'_j])$ for $j = 1, \ldots, m$ and all appropriate ground contexts c_j, so it follows from our assumption that $\theta^*(c[t]) = \theta^*(c[t'])$ for all ground contexts c. Let c be any general context; by Lemma 17 we have a ground context l such that $\theta^*(c[t]) = \theta^*(c[t'])$ follows from $\theta^*(l[t]) = \theta^*(l[t'])$; this latter follows from the fact that l is a ground context. This shows the 'only if' direction of the proposition; the proof of the other direction is similar. □

Similar reasoning shows that behavioral satisfaction defined over just local contexts is also equivalent; one key point is that the above proof only requires the context l given by Lemma 17 to be ground, whereas the lemma states that l is ground and local. We leave the (straightforward) details to the reader:

Proposition 19: A conditional equation $(\forall X)\ t = t'$ **if** $t_1 = t'_1, \ldots, t_m = t'_m$ is behaviorally satisfied by a hidden algebra A iff for all $\theta : X \to A$, whenever $\theta^*(c_j[t_j]) = \theta^*(c_j[t'_j])$ for $j = 1, \ldots, m$ and for all appropriate local contexts c_j, then $\theta^*(c[t]) = \theta^*(c[t'])$ for all appropriate local contexts c. Moreover, the same holds if only local ground contexts are used. □

It is useful to consider when two elements of an algebra behave the same under all experiments, which is a semantic notion of behavioral equivalence. We give a general definition that will allow us to prove behavioral satisfaction of equations by considering only contexts built from a given subsignature:

Definition 20: Given a hidden signature Σ with hidden subsignature $\Delta \subseteq \Sigma$, and given a hidden Σ-algebra A, then two elements $a, a' \in A_s$ are **behaviorally Δ-equivalent** iff $A_c(a) = A_c(a')$ for all appropriate local contexts $c \in L_\Delta[z]$ built from operations in Δ; in this case, we write $a \equiv_{\Delta,s} a'$, or just $a \equiv_\Delta a'$ if s is clear. We may call behavioral Σ-equivalence just **behavioral equivalence**. When we want to emphasize that behavioral equivalence is defined on a particular algebra A, we write $\equiv_A$ instead of $\equiv_\Sigma$. □

Fact 21: Given $\Delta \subseteq \Sigma$ and a hidden Σ-algebra A, then:

(1) for $a, a' \in A_v$ with v visible, $a \equiv_\Delta a'$ iff $a = a'$;

(2) $\equiv_\Delta$ is a Δ-congruence on A.

Proof: For (1), if $a = a'$, then $A_c(a) = A_c(a')$ for all contexts c. Conversely, if $a \equiv_\Delta a'$, then we can choose the 'empty' context z for sort v to get $A_z(a) = A_z(a')$, that is, $a = a'$.

For (2), $\equiv_\Delta$ is clearly an equivalence relation, so we need only show the congruence property, that for each operation $\sigma \in \Delta$, whenever $a_i \equiv_\Delta a'_i$ for $i = 1, \ldots, n$ we have

$$A_\sigma(a_1, ..., a_n) \equiv_\Delta A_\sigma(a'_1, ..., a'_n) \ .$$

If all of the a_i are visible, then this simply says that applying σ to equal arguments gives equal results, which is obvious. Otherwise, exactly one of the a_i is of hidden sort; for simplicity, let us assume it is a_1. Then for $i = 2, \ldots, n$ we have $a_i = a'_i$, and for any local Δ-context c, we can define the local context $c' = c[\sigma(z, a_2, \ldots, a_n)]$. Because $a_1 \equiv_\Delta a'_1$, we have

$$A_c(A_\sigma(a_1, ..., a_n)) = A_{c'}(a_1) = A_{c'}(a'_1) = A_c(A_\sigma(a'_1, ..., a'_n)) \ ,$$

as desired. $\square$

We will soon show how to use $\equiv_\Delta$ in proving the behavioral satisfaction of equations, but first we state a property needed in later sections.

Lemma 22: Given $\Delta \subseteq \Sigma$ and a hidden Σ-homomorphism $f\colon A \to B$, then for any a, a' in A, $a \equiv_\Delta a'$ in A iff $f(a) \equiv_\Delta f(a')$ in B.

Proof: By definition, $a \equiv_\Delta a'$ is equivalent to $A_c(a) = A_c(a')$ for all $c \in L_\Delta[z]$, which is equivalent to $f(A_c(a)) = f(A_c(a'))$ for all $c \in L_\Delta[z]$, because f is the identity on visible sorts. Now because f is a homomorphism, this in turn is equivalent to $B_c(f(a)) = B_c(f(a'))$ for all $c \in L_\Delta[z]$, which is by definition $f(a) \equiv_\Delta f(a')$. $\square$

Our statement of a fundamental result justifying several key techniques for proving behavioral satisfaction [26, 48, 31] uses the following:

Definition 23: Given $\Delta \subseteq \Sigma$, a **behavioral Δ-congruence** on a hidden Σ-algebra A is a Δ-congruence $\equiv$ which is equality on visible sorts, i.e., for $v \in V$ and $a, a' \in A_v = D_v$, we have $a \equiv_v a'$ iff $a = a'$. $\square$

The naturalness of this definition is brought out by reformulating the statement that $\equiv$ is equality on visible sorts as $\equiv\restriction_\Psi = 1_D$, in analogy with the definition of hidden homomorphism. Now the result:

Theorem 24: (Principle of Coinduction) Behavioral Δ-equivalence is the greatest behavioral Δ-congruence[1].

Proof: Fact 21 says that $\equiv_\Delta$ is a behavioral Δ-congruence. Suppose that R is a behavioral Δ-congruence and that $a \ R_s \ a'$. Then because R is a Δ-congruence, we have $A_c(a) \ R_v \ A_c(a')$ for any Δ-context c of appropriate sort; because R is equality on visible sorts, this implies that $A_c(a) = A_c(a')$ for any Δ-context c, i.e., that $a \equiv_\Delta a'$ as desired. $\square$

This implies that two states can be shown Δ-equivalent by finding *any* Δ-congruence that relates them. In [29] we call this proof technique **hidden coinduction**. Many specifications have a natural distinction between their 'generators' (or 'constructors') and 'destructors' (or 'selectors'), that can be exploited in coinduction proofs; in the jargon of the object paradigm, these are called 'methods' and 'attributes,' respectively. An example is given below; but first, we spell out how to use subsignatures in coinduction proofs. Note that $\Delta \subseteq \Sigma$ implies that $\equiv_\Sigma \ \subseteq \equiv_\Delta$ for any hidden Σ-algebra A; moreover, if $\equiv_\Delta$ is preserved by the other operations in the signature, then $\equiv_\Delta = \equiv_\Sigma$.

Proposition 25: Given $\Sigma = , \cup \Delta$ and a hidden Σ-algebra A, if $\equiv_\Delta$ is a , -congruence on A, then $\equiv_\Delta = \equiv_\Sigma$ on A.

Proof: We have already noted that $\equiv_\Delta \supseteq \equiv_\Sigma$, so it suffices to show $\equiv_\Delta \subseteq \equiv_\Sigma$. Since $\equiv_\Delta$ is a Δ-congruence, if it is also a , -congruence, then it is a Σ-congruence because $\Sigma = , \cup \Delta$. Since Theorem 24 says that $\equiv_\Sigma$ is the greatest behavioral Σ-congruence, we have $\equiv_\Delta \ \subseteq \ \equiv_\Sigma$ as desired. $\square$

This result can greatly simplify proofs of behavioral equivalence: instead of checking equality in all Σ-contexts, we need only check equality for all Δ-contexts. This latter proof obligation can be shown by induction on the structure of Δ-contexts; because Δ is a subsignature of Σ, there will be fewer cases to consider in the induction steps. But because this kind of "context induction" can be very tedious [16], it is fortunate that it is unnecessary, as illustrated in the following:

[1] This beautiful formulation arose in a conversation between Grant Malcolm and Rolf Hennicker in 1996, for the special case where $\Delta = \Sigma$.

Example 26: Given the `FLAG` theory in Example 12, define $f \simeq f'$ iff `up?` f =`up?` f' for flags f, f', and define $d \simeq d'$ iff $d = d'$ for data values d, d'. Then it is easy to check that $f \simeq f'$ implies `up` $f \simeq$ `up` f' and `dn` $f \simeq$ `dn` f' and `rev` $f \simeq$ `rev` f', and of course `up?` $f \simeq$ `up?` f'. So $\simeq$ is a behavioral congruence.

Therefore to show that every `FLAG`-algebra satisfies the equation (∀`F` : `Flag`) `rev rev F` = `F`, we can just show that `up? rev rev F` = `up? F`, which follows by simple equational reasoning, since `up? rev rev F` = `not(not(up? F))`.

It is easy to do this mechanically using OBJ3. We first set up the proof by opening `FLAG` and adding the necessary assumptions; here `R` represents the relation $\simeq$ (we omit its definition on visible sorts, because this is not used in the proof):

```
openr FLAG .
op _R_ : Flag Flag -> Bool .
var F1 F2 : Flag .
eq F1 R F2 = (up? F1 == up? F2) .
ops f1 f2 : -> Flag .
close
```

Here are the reductions showing that `R` is a behavioral congruence:

```
open .
eq up? f1 = up? f2 .
red (up f1) R (up f2) .     ***> should be: true
red (dn f1) R (dn f2) .     ***> should be: true
red (rev f1) R (rev f2) .   ***> should be: true
close
```

Finally, we show that all `FLAG`-algebras behaviorally satisfy the equation

(∀ `F` : `Flag`) `rev rev F` = `F`

with the following:

```
red (rev rev f1) R f1 .     ***> should be: true
```

All the above code runs in OBJ3, and gives `true` for each reduction, provided the following lemma about the Booleans is added somewhere:

```
eq not not B = B .
```

where `B` is a Boolean variable. □

3.2 More Satisfaction

This short section contains some results about behavioral satisfaction that are needed later.

Lemma 27: A hidden algebra A behaviorally satisfies $(\forall X)\, t = t'$ **if** $t_1 = t'_1, \ldots, t_m = t'_m$ iff for every assignment $\theta\colon X \to A$, whenever $\theta^*(t_j) \equiv_A \theta^*(t'_j)$ for $j = 1, ..., m$, then $\theta^*(t) \equiv_A \theta^*(t')$.

Proof: A behaviorally satisfies the equation iff for every $\theta\colon X \to A$, whenever $\theta^*(c[t_j]) = \theta^*(c[t'_j])$ for all appropriate c and for $j = 1, \ldots, m$, then also $\theta^*(c[t]) = \theta^*(c[t'])$ for all appropriate contexts c. Therefore by Lemma 16, A behaviorally satisfies the equation iff for all $\theta\colon X \to A$, we have $\theta^*(t) \equiv_A \theta^*(t')$ whenever $\theta^*(t_j) \equiv_A \theta^*(t'_j)$ for all $j = 1, \ldots, m$. □

Corollary 28: Given a hidden Σ-algebra A and a conditional Σ-equation all of whose terms have visible sort, then A satisfies the equation iff it behaviorally satisfies the equation.

Proof: This follows from Lemma 27 and (1) of Fact 21. □

Corollary 29: If a hidden Σ-algebra A satisfies a conditional Σ-equation all of whose conditions have visible sort, then it behaviorally satisfies that equation.

Proof: Suppose A satisfies an equation $(\forall X)\ t = t'$ **if** $t_1 = t'_1, \ldots, t_m = t'_m$, where each t_i and t'_i is of visible sort. If $\theta\colon X \to A$ is such that $\theta^*(t_i) \equiv_A \theta^*(t'_i)$ for $i = 1, \ldots, m$, then because behavioral equivalence is equality on visible sorts, we have $\theta^*(t_i) = \theta^*(t'_i)$, so $\theta^*(t) = \theta^*(t')$ and therefore $\theta^*(t) \equiv_A \theta^*(t')$, which shows that A behaviorally satisfies the equation. □

The following shows that Corollary 29 fails for conditional equations with hidden sorted conditions:

Example 30: Let Σ be the hidden signature with: hidden sort h and visible sort v; four constants a, b, c, d of sort h; and a function symbol $f\colon h \to v$. Also, let D_v be $\{0, 1\}$; and let A be the Σ-algebra with hidden carrier containing distinct constants A_a, A_b, A_c, A_d satisfying the equations

```
eq f(a) = 0 .
eq f(b) = 1 .
eq f(c) = f(d) .
```

Then A vacuously satisfies the conditional equation

$$(\forall \emptyset)\ a = b \ \textbf{ if } \ c = d \ ,$$

because $A_c \neq A_d$. However, A does not behaviorally satisfy this equation, because $c \equiv_A d$ but $a \not\equiv_A b$. □

3.3 Existence of Models

Unlike classical algebraic specification for abstract data values, where any specification has models (both initial and final), it is very easy to write behavioral theories that have *no* models. This is because adding equations reduces the class of models, and if the equations confuse data items, then there are no models; in this sense, the fixed data algebra D acts as a constraint on the possible models. Example 30 showed that not all hidden specifications have models, because if the conditional equation is added to the specification, then any model must behaviorally satisfy $a = b$, which implies $0 = 1$, a contradiction, since $0, 1$ are distinct elements of D. This section gives sufficient conditions for a theory to have at least one model, and also gives conditions for the existence of initial and final models.

3.3.1 Consistency

We first consider some examples that bring out the difficulties of showing consistency of hidden theories, after which we introduce some further definitions, give a necessary condition for consistency, and then some sufficient conditions. We seek conditions that are as syntactic as possible, so as to facilitate automatic verification of consistency. We thank Grigore Roşu for help with debugging and improving results in [32] for this section; however, he is not responsible for the current proof of Theorem 40, nor for any errors there or elsewhere.

Example 31: Consider a hidden theory having only one hidden sort h and one equation $(\forall \emptyset)\ 0 = 1$, where the data algebra consists of the two element set $\{0, 1\}$ with no operations. This hidden theory clearly has no models. But notice that if the equation is replaced by $(\forall x : h)\ 0 = 1$ where x is a variable of hidden sort h, then because there are no hidden constants of sort h, it would admit exactly one model, namely that with the carrier of sort h empty. □

A lesson of the above example is that to avoid inconsistency, one must avoid data conflicts. Since hidden models with empty carriers have no practical value, we exclude them in the following:

Definition 32: A hidden theory is **consistent** iff it has a model with all carriers non-empty. □

Sometimes inconsistency can involve properties of D, not just direct data conflicts. One simple example is the equation $(\forall \emptyset)\ 0 + 0 = 1$. This motivates the following:

Definition 33: Let $D^\star$ denote the set of all ground Ψ-equations of the form $(\forall \emptyset)\ t = d$ that are satisfied by D with $d \in D$. Then a set of Σ-equations E is D-**safe** iff for any $d, d' \in D$, if $E \cup D^\star \models_\Sigma (\forall \emptyset)\ d = d'$ then $d = d'$. □

Fact 34: If e is a Ψ-equation, then $D^\star \models_\Psi e$ iff $D \models_\Psi e$.

Proof: Since $D \models D^\star$, the forward direction is clear. For the converse, it suffices to establish the result for ground equations, where it is true by the definition of $D^\star$. □

For the rest of this section suppose that $P = (\Sigma, E)$ is a hidden theory over a data algebra D with no empty carriers. The following gives a natural necessary condition for consistency:

Proposition 35: If P is consistent then E is D-safe.

Proof: Let M be a model of P with all carriers non-empty. For the sake of contradiction, suppose there are distinct $d, d' \in D$ such that $E \cup D^\star \models_\Sigma (\forall X)\ d = d'$. Because ordinary equational deduction is sound for behavioral satisfaction (by Proposition 58 below), we get $E \cup D^\star \models\!\equiv_\Sigma (\forall X)\ d = d'$, and because $M \models\!\equiv_\Sigma E \cup D^\star$, we then get $M \models\!\equiv_\Sigma (\forall X)\ d = d'$, which is a contradiction because $d \neq d'$ and no carriers of M are empty. □

However D-safety is not a sufficient condition for consistency, because conflicts more subtle than in Example 31 can arise:

Example 36: Consider the hidden theory with D the natural numbers with addition, with one hidden sort h, one hidden constant c, one attribute $a\colon\ h \to v$, and just one equation in E, $(\forall \emptyset)\ 1 + a(c) = a(c)$. This theory is inconsistent because there is no natural number n for the value of $a(c)$ such that $1+n = n$. However, E is D-safe because there are no distinct natural numbers $n, m \in D$ such that $E \cup D^\star \models_\Sigma (\forall \emptyset)\ n = m$. To show this, it suffices to find a Σ-model M (which need not protect D) which satisfies E and $D^\star$ but does not satisfy $(\forall \emptyset)\ n = m$ for any distinct n, m. We let M have the natural numbers plus a new element ∞ as its carrier of sort v, with $n + \infty = \infty$ for any n and $\infty + \infty = \infty$, and with a one-element set $\{*\}$ as its carrier of sort h, with c interpreted as $*$ and $a(c)$ as ∞. □

The above example shows that using operations in Ψ (such as addition) on top of attributes in equations is dangerous, because it can induce conflicts on data. The following example shows that data conflicts can appear even when the equations are D-safe and involve no operations in Ψ.

Example 37: Consider a hidden theory over the data algebra having carrier $\{0, 1\}$ of sort v with no operations, having one hidden sort h and one hidden constant x of sort h, two attributes $a\colon\ h \to v$ and $b\colon\ h\ v \to v$, and having three equations in E, $(\forall \emptyset)\ b(x, a(x)) = 0$, $(\forall \emptyset)\ b(x, 0) = 1$, and $(\forall \emptyset)\ b(x, 1) = 1$. This theory is inconsistent because if a hidden model existed, then because $a(x)$ must be either 0 or 1, the first equation and either the second or the third, would imply that the equation $(\forall \emptyset)\ 0 = 1$ is satisfied by that model. Notice also that it is not the case that $E \cup D^\star \models_\Sigma (\forall \emptyset)\ 0 = 1$ because as in Example 36, there are models satisfying $E \cup D^\star$ where $0 \neq 1$. □

Examples 36 and 37 suggest that it may be difficult to state consistency criteria for equations with non-local terms even if they are D-safe, as in the previous two examples. We will soon see (Theorem 40) that D-safety plus locality of equations is sufficient for consistency when the equations are non-conditional; in fact, Theorem 40 covers even conditional equations whose conditions contain only Ψ-terms. But first, we briefly consider some difficulties that can arise with conditional equations.

Example 38: Consider the hidden theory with one visible sort v, one hidden sort h, D the set $\{0, 1\}$, one attribute $a\colon\ h \to v$, four hidden constants x, x', y, y', and the equations $(\forall \emptyset)\ a(x) = 0$, $(\forall \emptyset)\ a(x') = 1$, $(\forall \emptyset)\ a(y) = a(y')$, and $(\forall \emptyset)\ x = x'$ **if** $y = y'$. This theory is inconsistent, because if it had a model A then $A_y \equiv A_{y'}$ (since there is only one experiment, $a(z)$), therefore the condition is satisfied, and so A_x is behaviorally equivalent to $A_{x'}$; but this would imply that $A_x, A_{x'}$ give the same value under the experiment $a(z)$, so that $0 = 1$. It is easy to check that these equations are D-safe and involve only local terms. □

One may suspect that the inconsistency above occurred because of the hidden condition of the equation. The next example shows that inconsistencies can appear even if the conditions are visible and the terms in equations are local.

Example 39: Consider a hidden theory with one visible sort v and one hidden sort h, $D = \{0, 1\}$, one hidden constant x, one attribute $a\colon\ h \to v$, and two conditional equations, $(\forall \emptyset)\ a(x) = 0$ **if** $a(x) = 1$ and $(\forall \emptyset)\ a(x) = 1$ **if** $a(x) = 0$. Because one of the two conditions must be satisfied in any model, it follows that $0 = 1$, so that this theory is inconsistent. It can also be shown that the equations are D-safe; we encourage the reader to find an appropriate model for this purpose. □

Examples 38 and 39 indicate difficulties with conditional equations where the conditions contain operations not in Ψ, and the last four examples together motivate the following, recalling that locality implies that all the conditions in equations are visible:

Theorem 40: A hidden specification $P = (H, \Sigma, E)$ is consistent if E is D-safe and local.

Proof: We demonstrate the consistency of P by exhibiting a model M for P. Without loss of generality, we may assume that Σ contains a constant of each hidden sort, because if it didn't, we could add such constants to obtain a larger signature over which E is still D-safe and local, so that a model of it also yields a (reduct) model of the original Σ with all carriers non-empty, as is required for consistency.

Letting d_v be some arbitrary but fixed element in D_v for each visible sort v, we define M as follows:

- $M{\restriction_\Psi} = D$;
- for each hidden sort h, let $M_h = L_{\Sigma,h}$, the local ground terms of sort h;
- for each method $\sigma\colon\ h\ w \to h'$, define $M_\sigma\colon\ M_h \times D_w \to M_{h'}$ by $M_\sigma(l, d) = \sigma(l, d)$ for $l \in M_h$ and $d \in D_w$ (noting that $M_\sigma(l, d)$ is local if l is); and
- for each attribute $\sigma\colon\ h\ w \to v$, define $M_\sigma\colon\ M_h \times D_w \to D_v$ for $l \in M_h$ and $d \in D_w$ by

$$M_\sigma(l, d) = \begin{cases} d' & \text{when there is some } d' \in D \text{ such that } E \cup D^\star \models_\Sigma (\forall \emptyset)\ \sigma(l, d) = d' \\ d_v & \text{otherwise .} \end{cases}$$

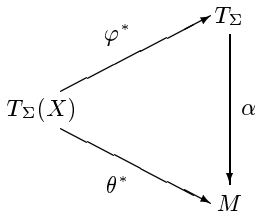
Since E is D-safe, if any d' as above exists, then it is unique. Thus M is a well-defined hidden Σ-algebra.

We first show that for any local terms l and l' of visible sort v,

$$M_l = M_{l'} \quad \text{if} \quad E \cup D^\star \models (\forall \emptyset)\ l = l' \ . \tag{1}$$

We prove this by the following case analysis, assuming $E \cup D^\star \models (\forall \emptyset)\ l = l'$:

- If l and l' are both Ψ-terms, then $M_l = D_l = D_{l'} = M_{l'}$.
- If l is not a Ψ-term, then $l = \sigma(t, d)$ for some $\sigma\colon\ h\,w \to v$, with t a local term of hidden sort h and $d \in D_w$ (actually, d may consist of Ψ-terms, but since we are concerned with equality under the equations in $D^\star$, we may safely consider Ψ-terms to be equal to their values in D). If l' is a Ψ-term, then there is $d' \in D_v$ with $D^\star \models (\forall \emptyset)\ l' = d'$ (i.e., $d' = D_{l'} = M_{l'}$). It follows that $E \cup D^\star \models (\forall \emptyset)\ \sigma(t, d) = d'$, so by the definition of M we have $M_l = M_\sigma(t, d) = d' = M_{l'}$.
- If neither l nor l' are Ψ-terms, then we have $l = \sigma(t, d)$ and $l' = \sigma'(t', d')$ for some σ, σ', etc. For any $d'' \in D_v$ we have $E \cup D^\star \models (\forall \emptyset)\ \sigma(t, d) = d''$ iff $E \cup D^\star \models (\forall \emptyset)\ \sigma'(t', d') = d''$. It now follows from the definition of M that if there is some such d'', then $M_l = M_\sigma(t, d) = d'' = M_{\sigma'}(t', d') = M_{l'}$, and if there is no such d'', then $M_l = d_v = M_{l'}$.



To show that M behaviorally satisfies all equations in E, consider $e \in E$ of the form $(\forall X)\, t = t'$ **if** $t_1 = t'_1, \ldots, t_m = t'_m$ and let $\theta\colon\ X \to M$ be such that $\theta^*(t_i) = \theta^*(t'_i)$ for $i = 1, \ldots, m$ (the hypothesis in Definition 11 reduces to this because all the t_i, t'_i are $\Psi(X)$-terms). We will show $M \models\!\!\models e$ by showing that $\theta^*(c[t]) = \theta^*(c[t'])$ for any appropriate local ground context c (see Proposition 19).

Let α be the unique Σ-morphism $T_\Sigma \to M$, noting that it maps $t \in T_\Sigma$ to M_t (i.e., $\alpha(t) = M_t$ for $t \in T_\Sigma$), and in particular, maps $t \in T_\Psi$ to D_t. Then α is surjective because $M_l = l$ if $l \in M$ (as can be shown by induction on the structure of local ground terms). Hence there exists $\varphi\colon\ X \to T_\Sigma$ such that $\theta = \alpha \circ \varphi$; in fact, we can actually choose $\varphi(x) = \theta(x)$, since $M \subseteq T_\Sigma$. Therefore $\theta^* = \alpha \circ \varphi^*$, and in fact, $\theta^*(t) = \varphi^*(t)$, from which it follows that $\varphi^*(t)$ is always local, and that $M_{\varphi^*(t)} = \theta^*(t)$ if t is local. Now

$$D \models_\Sigma (\forall \emptyset)\ \varphi^*(t_i) = \varphi^*(t'_i)$$

for $i = 1, \ldots, m$, because $\theta^*(t_i) = \theta^*(t'_i)$ by assumption. Therefore by Fact 34,

$$D^\star \models_\Sigma (\forall \emptyset)\ \varphi^*(t_i) = \varphi^*(t'_i)$$

for $i = 1, \ldots, m$, so that also

$$E \cup D^\star \models_\Sigma (\forall \emptyset)\ \varphi^*(t_i) = \varphi^*(t'_i)$$

and therefore

$$E \cup D^\star \models_\Sigma (\forall \emptyset)\ \varphi^*(t) = \varphi^*(t')\ ,$$

by applying $e \in E$. Hence, for any appropriate local ground context c,

$$E \cup D^\star \models_\Sigma (\forall \emptyset)\ c[\varphi^*(t)] = c[\varphi^*(t')]\ ,$$

so that by Lemma 16,

$$E \cup D^\star \models_\Sigma (\forall \emptyset)\ \varphi^*(c[t]) = \varphi^*(c[t'])\ ,$$

which by condition (1) implies $M_{\varphi^*(c[t])} = M_{\varphi^*(c[t'])}$, which gives $\theta^*(c[t]) = \theta^*(c[t'])$, as desired. □

Although all the examples in this paper satisfy the hypotheses of Theorem 40 for consistency, unfortunately this is not machine checkable because D-safety is a semantic condition. Therefore we would like to have good machine checkable sufficient conditions for D-safety. One such is given in Proposition 42 below, but first we need the following:

Definition 41: A Σ-term rewriting system R is **D-confluent** iff $R \cup R_{D^\star}$ is confluent, where $R_{D^\star} = \{t \to d \mid D \models_\Psi (\forall \emptyset)\ t = d\}$ is the Ψ-rewriting system associated to $D^\star$. □

So a Σ-term rewriting system is D-confluent iff it is confluent modulo evaluations of ground Ψ-terms in D. D-confluence seems very natural for reasoning over a fixed data universe, and we believe that confluence criteria like orthogonality can be adapted to it.

Proposition 42: If E can be oriented as a D-confluent Σ-term rewriting system with no rule having some $d \in D$ as its left side, then E is D-safe.

Proof: For the sake of contradiction, suppose E is not D-safe, i.e. that there are distinct $d, d' \in D$ such that $E \cup D^\star \models_\Sigma (\forall X)\ d = d'$. Since E can be oriented as a D-confluent Σ-term rewriting system, say R_E, completeness of equational reasoning implies $d\ (\to \cup \leftarrow)^*\ d'$, where $\to$ is the rewriting relation induced by the Σ-term rewriting system $R_E \cup R_{D^\star}$ and $\leftarrow$ is its converse. Since $R_E \cup R_{D^\star}$ is confluent, we get $d\ (\to^*; \leftarrow^*)\ d'$, a contradiction since there is no rule having d or d' as left side in R_E, and the only rules in $R_{D^\star}$ having d or d' as their left sides are $d \to d$ and $d' \to d'$. □

3.3.2 Initial and Final Models

We turn now to necessary and sufficient conditions for the existence of initial and final models for hidden theories. We give the constructions here, while subsequent sections examine their logical properties.

Definition 43: Given a hidden theory $P = (H, \Sigma, E)$, a ground Σ-term t is **defined** iff for every context c (of appropriate sort), there is some $d \in D$ such that $E \models\!\!\equiv c[t] = d$; otherwise, we say t is **undefined**. P is **lexic** iff all ground Σ-terms are defined. $\square$

We note that L_Σ is "almost" a hidden Σ-algebra, in that $L_{\Sigma,v} = D_v$ for $v \in V$, and that any Ψ-homomorphism $f\colon T_\Sigma|_V \to D$ gives rise to a hidden algebra that we denote L_f, with $L_{f,h} = L_{\Sigma,h}$ for $h \in H$, having methods interpreted the obvious way as term building operations, and having attributes $\sigma \in \Sigma_{w,v}$ interpreted by

$$L_{f,\sigma}(\overline{x}) = f(\sigma(\overline{x}))$$

for arguments $\overline{x} \in L_{f,w}$. In particular, every hidden Σ-algebra A gives rise to a Σ-homomorphism $T_\Sigma \to A$ interpreting visible terms as data values, which in turn produces an f as above by restricting to V. Very similar considerations apply to L_A in the following:

Definition 44: For any hidden Σ-algebra A, write L_A for the hidden Σ-algebra L_f induced by the homomorphism $f\colon T_\Sigma \to A$, and l_A for the hidden Σ-homomorphism $L_A \to A$ defined by $l_A(t) = f(t)$. $\square$

Because l_A is the Σ-homomorphism f restricted to local terms, we have

Fact 45: l_A is the unique hidden homomorphism $L_A \to A$. $\square$

For lexic theories, all the models L_A are identical, which is a key step in proving the following:

Theorem 46: (Hidden Initiality) A hidden theory P has an initial model I_P iff P is consistent and lexic.

Proof: If a hidden theory (Σ, E) is consistent and lexic, then for every visible ground term t there is a unique d_t in D with $E \models\!\!\equiv (\forall \emptyset)\ t = d_t$. Moreover, for any hidden model A of E, the homomorphism $f\colon T_\Sigma \to A$ necessarily agrees with the mapping $t \mapsto d_t$, and since it is this mapping that defines L_A, it follows that $L_A = L_B$ for any two models A and B. Because E is consistent, there is at least one model, say A, so we have L_A and a unique hidden homomorphism $l_A\colon L_A \to A$. By Lemma 50 below, L_A behaviorally satisfies all the equations that A does, so that $L_A \models\!\!\equiv E$. Moreover, for any model B, there is a unique hidden homomorphism $l_B\colon L_A = L_B \to B$. Therefore we can take $I_P = L_A$ as the initial hidden algebra.

Conversely, if there is an initial model, then E is consistent. For every visible term t, there is a data value $f(t)$ given by the homomorphism f from T_Σ to the initial model. Moreover, for any other model A with homomorphism $g\colon T_\Sigma \to A$, it follows that $g(t) = h(f(t)) = f(t)$, where h is the unique hidden Σ-homomorphism from the initial model to A, which shows that $A \models\!\!\equiv (\forall \emptyset)\ t = f(t)$, and since A is an arbitrary model, we conclude that every ground term t is defined by a data value $f(t)$ with $E \models\!\!\equiv (\forall \emptyset)\ t = f(t)$. $\square$

Initial models are less important for the hidden paradigm than they are for initial algebra semantics [39] or the more general initial model semantics [34]. Final algebras come closer to that role, in that they capture many abstract properties of the state space. We will show that final models exist exactly when initial models do, provided each equation contains at most one hidden variable; however, final models are less common in the recent more general hidden framework where operations may have more than one hidden argument, which has motivated ways to obtain similar results without using final models [38, 58, 25].

Definition 47: Let $\Sigma^\diamond$ denote the signature obtained from Σ by forgetting all generalized constants in Σ; i.e., $\Sigma^\diamond_{w,s} = \emptyset$ if $w \in V^*$ and $s \in H$, and $\Sigma^\diamond_{w,s} = \Sigma_{w,s}$ otherwise. For a hidden Σ-algebra A, we will write $A^\diamond$ for the reduct $A|_{\Sigma^\diamond}$. Now let $F_{\Sigma^\diamond}$ be the hidden $\Sigma^\diamond$-algebra defined by the following "magical formula"

$$F_{\Sigma^\diamond,h} = \prod_{v \in V} [L_\Sigma[z_h]_v \to D_v]$$

for hidden sorts $h \in H$. As with all hidden signatures in this paper, the methods and attributes of $\Sigma^\diamond$ have only one argument of hidden sort; for simplicity, we assume that this is always the first argument. $F_{\Sigma^\diamond}$ interprets the operations of $\Sigma^\diamond$ as follows:

- For methods $\sigma \in \Sigma_{hw,h'}$, $p \in F_{\Sigma^\diamond,h}$, $d \in D_w$, and $c \in L_\Sigma[z_{h'}]_v$, let $(F_{\Sigma^\diamond})_\sigma(p,d)(c) = p_v(c[\sigma(z_{h'},d)])$.
- For attributes $\sigma \in \Sigma_{hw,v}$, $p \in F_{\Sigma^\diamond,h}$, $d \in D_w$, let $(F_{\Sigma^\diamond})_\sigma(p,d) = p_v(\sigma(z_h,d))$.

□

Proposition 48: $F_{\Sigma^\diamond}$ is a final hidden $\Sigma^\diamond$-algebra.

Proof: Given a hidden $\Sigma^\diamond$-algebra A, the unique hidden homomorphism $A^\diamond \to F_\Sigma$ takes $a \in A_h$ to the family (over $v \in V$) of mappings $L_\Sigma[z_h]_v \to D_v$ sending $c \in L_\Sigma[z_h]_v$ to $A_c(a)$. □

The unique homomorphism $A^\diamond \to F_\Sigma$ taking a hidden state to all its observable behaviors can be thought of as evaluating all attributes for all states that can be reached from the given state. It can also be shown that two elements of a hidden algebra are behaviorally equivalent iff their images in the final algebra are equal [46]. This prepares us for

Theorem 49: If the equations in a hidden theory $P = (H, \Sigma, E)$ have at most one variable of hidden sort, then P has a final model, denoted F_P, iff it is consistent and lexic.

Proof: Suppose P is consistent and lexic, and for any P-algebra A let $\varphi\colon A^\diamond \to F_{\Sigma^\diamond}$ be the unique hidden $\Sigma^\diamond$-homomorphism to the final algebra $F_{\Sigma^\diamond}$, made a hidden Σ-algebra by interpreting generalized constants $\sigma \in \Sigma_{w,h}$ by $(F_{\Sigma^\diamond})_\sigma(d) = \varphi(A_\sigma(d))$ for all $d \in D_w$; note that φ is a hidden Σ-homomorphism. Let F_A be the image of φ, i.e., factor φ as the composition of surjective $\varphi_0\colon A^\diamond \to F_A$ and inclusive $\varphi_1\colon F_A \hookrightarrow F_{\Sigma^\diamond}$. Because φ_0 is surjective, Lemma 54 below implies that $F_A \models\!\!\equiv E$. Now let F_P be the greatest subalgebra of $F_{\Sigma^\diamond}$ that behaviorally satisfies E; in fact, F_P is the union of all the images F_A for each hidden P-algebra A. For any equation in E with variables X, because at most one variable in X is of hidden sort, any assignment $\theta\colon X \to F_P$ is an assignment $\theta\colon X \to F_A$ for some A, and so $F_P \models\!\!\equiv E$. For any P-algebra A, we have already noted that F_A is a subalgebra of $F_{\Sigma^\diamond}$ that behaviorally satisfies E; therefore it is contained in F_P, which shows that the domain of φ lies in F_P, which is therefore final. This concludes the 'if' direction of the proof. The converse is like that of Theorem 46. □

4 Hidden Algebras and Hidden Proofs

This section gives some basic results in hidden model theory, including the soundness of equational reasoning for proving behavioral satisfaction, as well as some more sophisticated proof techniques. We start with a very basic property of hidden homomorphisms:

Lemma 50: Given a hidden homomorphism $f\colon A \to B$ and an equation e, then $B \models\!\!\equiv e$ implies $A \models\!\!\equiv e$.

Proof: Let e be of the form $(\forall X)\ t = t'$ **if** $t_1 = t'_1, \ldots, t_m = t'_m$ and let $\theta\colon X \to A$ be such that $\theta^*(t_j) \equiv_A \theta^*(t'_j)$ for $j = 1, \ldots, m$. Then by Lemmas 1 and 22, we have $(f \circ \theta)^*(t_j) \equiv_B (f \circ \theta)^*(t'_j)$, so if $B \models\!\!\equiv e$ then $(f \circ \theta)^*(t) \equiv_B (f \circ \theta)^*(t')$. Again by Lemmas 1 and 22, this implies that $\theta^*(t) \equiv_A \theta^*(t')$, which proves that $A \models\!\!\equiv e$ if $B \models\!\!\equiv e$. □

Proposition 51: For any hidden theory P:

(1) an initial P-algebra behaviorally satisfies an equation iff some P-algebra behaviorally satisfies it;

(2) a final P-algebra behaviorally satisfies an equation iff all P-algebras behaviorally satisfy it.

Proof: Immediate from Lemma 50. □

Lemma 50 states that satisfaction of equations propagates backwards along hidden homomorphisms; we now show that satisfaction also propagates forwards for ground equations, as well as along surjective homomorphisms.

Lemma 52: Given a hidden homomorphism $f: A \to B$ and a ground equation e, if $A \models\!\equiv e$ then $B \models\!\equiv e$.

Proof: Suppose e has the form $(\forall \emptyset)\ t = t'$ **if** $t_1 = t'_1, \ldots, t_m = t'_m$ and that $A \models\!\equiv e$, i.e., if $g(t_j) \equiv_A g(t'_j)$ for $j = 1, \ldots, m$, then $g(t) \equiv_A g(t')$, where g is the unique homomorphism from T_Σ to A. If $h(t_j) \equiv_B h(t'_j)$ for $j = 1, \ldots, m$, where h is the unique homomorphism $T_\Sigma \to B$, then $f(g(t_j)) \equiv_B f(g(t'_j))$, because $h = f \circ g$; by Lemma 22 this implies $g(t_j) \equiv_A g(t'_j)$ for $j = 1, \ldots, m$, and because $A \models\!\equiv e$ we get $g(t) \equiv_A g(t')$. Now by Lemma 22 again, $f(g(t)) \equiv_B f(g(t'))$, i.e., $h(t) \equiv_B h(t')$, which shows that $B \models\!\equiv e$. □

Corollary 53: If $P = (\Sigma, E)$ is a hidden theory with initial hidden algebra I_P, and if e is a ground Σ-equation, then $I_P \models\!\equiv e$ iff all P-algebras behaviorally satisfy e. □

Lemma 54: Given a surjective hidden homomorphism $f: A \to B$ and an equation e, if $A \models\!\equiv e$ then $B \models\!\equiv e$.

Proof: Let e be $(\forall X)\ t = t'$ **if** $t_1 = t'_1, \ldots, t_m = t'_m$ and let $\theta: X \to B$ be such that $\theta^*(t_j) \equiv_B \theta^*(t'_j)$ for $j = 1, \ldots, m$. Because f is surjective, there is some $\eta: X \to A$ such that $\theta = f \circ \eta$; then $f(\eta^*(t_j)) \equiv_B f(\eta^*(t'_j))$ and so by Lemma 22, $\eta^*(t_j) \equiv_A \eta^*(t'_j)$ for $j = 1, \ldots, m$. Since $A \models\!\equiv e$, this implies $\eta^*(t) \equiv_A \eta^*(t')$, and so by Lemma 22 $f(\eta^*(t)) \equiv_B f(\eta^*(t'))$, i.e., $\theta^*(t) \equiv_B \theta^*(t')$, so that $B \models\!\equiv e$. □

Corollary 55: If $P = (\Sigma, E)$ is a hidden theory with initial hidden algebra I_P and e is a Σ-equation, then $I_P \models\!\equiv e$ iff all reachable hidden P-algebras behaviorally satisfy e.

Proof: A P-algebra A is reachable iff the unique homomorphism $h: T_\Sigma \to A$ is surjective; such an h factors through the unique hidden homomorphism $I_P \to A$, which is therefore also surjective, and now the 'only if' implication follows from Lemma 54. The 'if' direction follows from the fact that I_P is reachable. □

An immediate corollary to this and Proposition 51 is that for consistent, lexic hidden theories, the behavior of all reachable algebras is the same:

Corollary 56: If $P = (\Sigma, E)$ is a consistent, lexic hidden theory, and if e is a Σ-equation behaviorally satisfied by some P-algebra, then all reachable hidden P-algebras behaviorally satisfy e. □

4.1 Proving Behavioral Satisfaction

This section presents some techniques for proving behavioral satisfaction, particularly hidden coinduction. The next section gives a sample coinduction proof, but we begin by showing soundness of perhaps the most elementary technique for proving behavioral satisfaction, which is ordinary equational reasoning. But first:

Lemma 57: Given a hidden Σ-algebra A and a (possibly conditional) Σ-equation e, then $A/_{\equiv_\Sigma} \models e$ iff $A \models\!\equiv e$.

Proof: This is immediate from Lemma 27. □

Proposition 58: If an unconditional Σ-equation e is derivable with ordinary equational reasoning from a set of equations E, then all hidden (Σ, E)-algebras behaviorally satisfy e. □

This follows from the more general result below, because if $(\forall X)\ t = t'$ is derivable from E, then so are all the equations $(\forall X)\ c[t] = c[t']$ for appropriate contexts c, by the congruence rule of equational deduction.

Proposition 59: If for every appropriate context c, the Σ-equation $(\forall X)\ c[t] = c[t']$ is derivable by ordinary equational deduction from a set of equations E, then every hidden (Σ, E)-algebra behaviorally satisfies $(\forall X)\ t = t'$.

Proof: If $(\forall X)\ c[t] = c[t']$ is derivable from E for all c, then $E \models (\forall X)\ c[t] = c[t']$ for all c, so that for any hidden (Σ, E)-algebra A, if $A/_{\equiv_\Sigma} \models E$ then $A/_{\equiv_\Sigma} \models (\forall X)\ c[t] = c[t']$ for each c, which means that $A \models\!\equiv (\forall X)\ c[t] = c[t']$, i.e., that $A \models (\forall X)\ c[t] = c[t']$ for all c, since c is visible, which implies that $A \models\!\equiv (\forall X)\ t = t'$. □

However, ordinary equational reasoning is not complete for behavioral satisfaction: there are many hidden theories where many equations not derivable by equational deduction are behaviorally satisfied by all models (e.g., see Example 26). Proposition 59 justifies using induction over the structure of contexts to prove behavioral satisfaction; however this can be very complex, as illustrated by experiences reported in [16]. In fact, there cannot be any complete finite inference rule set for hidden satisfaction [5]. Results in Section 3.1 show that it is possible to restrict attention to contexts built from certain subsignatures $\Delta \subseteq \Sigma$. The proof in Example 26 can be thought of as a trivial induction on contexts: there is only one possible context, namely `up?` z. However, it is also an example of *hidden coinduction* [31], which involves showing that two elements are behaviorally equivalent by finding some behavioral congruence that relates them. This technique is justified by Theorem 24. Hidden coinduction proofs have three steps: first, a "candidate relation" is proposed; second, this relation is shown to be a behavioral congruence; and third, it is shown that the relation relates the two elements to be shown behaviorally equivalent. Example 26 actually illustrates a more general form of hidden coinduction proof, since the declarations

```
op _R_ : Flag Flag -> Bool .
var F1 F2 : Flag .
eq F1 R F2 = (up? F1 == up? F2) .
```

define a candidate relation for all `FLAG`-models. The reductions in the example use rewriting to show that this candidate relation is a behavioral congruence. Since Proposition 58 shows that equational deduction is sound, it follows that the candidate really is a behavioral congruence for all `FLAG`-models. Thus, one useful technique for finding candidate relations for coinductive proofs is to use behavioral Δ-equivalence for some subsignature Δ; another technique is given in Proposition 61 below. But first we introduce:

Notation 60: Suppose $\Sigma = , \cup \Delta$ where , contains no visible operations (i.e., no attributes), and let $R \subseteq A \times A$ be a relation on a Σ-algebra A such that R is the equality relation on visible sorts. Then let R^{Γ} denote the least behavioral , -congruence extending both R and $\equiv_{\Sigma}$. □

This assumes that the least behavioral , -congruence extending R exists; in fact, we can take R^{Γ} to be the least , -congruence extending R, because since , contains no visible operations, the least , -congruence extending R and $\equiv_{\Sigma}$ cannot relate any new data items, and therefore must be the equality relation on visible sorts. This means that the least , -congruence extending R is in fact a behavioral , -congruence.

Coinduction proofs can be considered to generalize the bisimilarity proofs used in process algebra [46]: since bisimilarity is the greatest bisimulation [52, 53], one can prove that two states are bisimilar by exhibiting any bisimulation that relates them. Similarly, behavioral equivalence of two states can be shown by exhibiting any behavioral congruence that relates them. A common technique for bisimilarity proofs is to extend some relation that relates the two states to a bisimulation. A similar technique works for the more general method of coinduction, and the next subsection uses Proposition 61 below for this purpose.

Proposition 61: Let $\Sigma = , \cup \Delta$ and $R \subseteq A \times A$ be as in Notation 60 above. If

(1) $(A_\delta(a,d), A_\delta(a',d)) \in R^{\Gamma}_s$ for every $\delta \in \Delta_{hw,s}$, $(a,a') \in R_h$ and $d \in D_w$, and if

(2) $(A_\delta(A_\gamma(a,e),d), A_\delta(A_\gamma(a',e),d)) \in R^{\Gamma}_s$ for every $\delta \in \Delta_{hw,s}$, $\gamma \in , _{h'w',h}$, $(a,a') \in R^{\Gamma}_{h'}$, $d \in D_w$ and $e \in D_{w'}$,

then $R^{\Gamma} \subseteq \equiv_{\Sigma}$.

Proof: Let ΔR^{Γ} be the following relation,

$$\{(a,a') \in R^{\Gamma} \mid (A_\delta(a,d), A_\delta(a',d)) \in R^{\Gamma}_s \text{ for every } \delta \in \Delta_{hw,s} \text{ and } d \in D_w\} \ .$$

It is straightforward to show that ΔR^{Γ} is an equivalence relation. It extends $\equiv_{\Sigma}$, because if $a \equiv_{\Sigma} a'$, then $A_\delta(a,d) \equiv_{\Sigma} A_\delta(a',d)$, so $(A_\delta(a,d), A_\delta(a',d)) \in R^{\Gamma}$ and therefore $(a,a') \in \Delta R^{\Gamma}$. Moreover, (1) says that ΔR^{Γ} extends R, and (2) implies it is a , -congruence. Since R^{Γ} is the least , -congruence extending R and $\equiv_{\Sigma}$, we have $R^{\Gamma} \subseteq \Delta R^{\Gamma}$, and therefore $R^{\Gamma} = \Delta R^{\Gamma}$. This means that R^{Γ} is a Δ-congruence, because by definition of ΔR^{Γ}, if $(a,a') \in R^{\Gamma}_h$ then $(A_\delta(a,d), A_\delta(a',d)) \in R^{\Gamma}_s$ for each δ in Δ. Therefore R^{Γ} is a behavioral Σ-congruence, and so $R_{\Gamma} \subseteq \equiv_{\Sigma}$ follows by Theorem 24. □

4.2 An Example

This section uses the above results to prove behavioral properties of the following hidden specification:

```
th STACK2 is sort Stack .
  pr DATA .
  op push : Nat Stack -> Stack .
  op empty : -> Stack .
  op empty' : -> Stack .
  op pop_ : Stack -> Stack .
  op top_ : Stack -> Nat .
  var S : Stack .  var N : Nat .
  eq pop push(N,S) = S .
  eq top push(N,S) = N .
  eq pop empty = empty .
  eq pop empty' = empty' .
  eq top empty  = 0 .
  eq top empty' = 0 .
endth
```

This is the same as the STACK specification in Example 13, except that there is a new hidden constant `empty'` with two new equations. The signature Ψ of data values is the signature of DATA, which we assume specifies at least the natural numbers, with a constant for each natural number, and we also assume that the fixed Ψ-algebra contains at least the usual algebra of natural numbers with the usual operations.

Proposition 62: STACK2 is consistent and lexic.

Proof: Since the equations of STACK2 are local, Theorem 40 says we need only check the conditions of Proposition 42, plus lexicality. We give only a sketch; standard techniques can be used to show D-confluence. For lexicality, a straightforward inductive argument shows that every term of sort Stack has a normal form which is a local term built from push, empty, empty', and terms of sort Nat, and we suppose that the equations of NAT are such as to guarantee that these have data values as normal forms. Therefore applying top to a term of sort Stack always yields a data value. □

Now suppose we want to know if the equation

```
pop push(N,pop S) = pop pop push(N,S)
```

is behaviorally satisfied by all STACK2-algebras. This equation is derivable from the equations in STACK2 and the rules of inference in Definition 3 as follows:

`pop push(N,S) = S` is derivable by rule (0), so
`pop pop push(N,S) = pop S` is derivable by rule (4) and
`pop S = pop pop push(N,S)` is derivable by rule (2);
`pop S = pop S` is derivable by rule (1), and
`pop push(N,pop S) = pop S` is derivable by rule (5) so
`pop push(N,pop S) = pop pop push(N,S)` by rule (3).

Therefore Proposition 58 tells us that all STACK2-algebras behaviorally satisfy the equation. An easier way to show satisfaction of this equation is to use term rewriting, which shows that the normal forms of both sides of this equation are the same, namely `pop S`, as follows (using OBJ3)

```
red pop push(N,pop S) == pop pop push(N,S) .
```

so that, again by Proposition 58, the equation is behaviorally satisfied by all reachable STACK2-algebras.

In contrast, the following equation is less easy to prove behaviorally satisfied,

$(\forall \emptyset)$ `empty = empty'` ,

because `empty` and `empty'` are already in normal form, and are unequal, so that we cannot use equational deduction. We could use Proposition 59 and induction on contexts to show satisfaction in all models, by proving the family of equations $c[\texttt{empty}] = c[\texttt{empty}']$, where c is a context built from operator symbols in the signature of `STACK2`. But Proposition 61 gives a much more elegant proof. Essentially, we take the equation to be proved as coinduction hypothesis, i.e., we let R be the relation relating `empty` and `empty'`, and show that the least behavioral , -congruence extending this relation and $\equiv_\Sigma$ satisfies the conditions of Proposition 61, where , contains just `push`, `empty` and `empty'`, and Δ contains `top` and `pop`, as in Section 3.1. In fact, we show that any such , -congruence satisfies the conditions of Proposition 61. Here is how the proof would look in OBJ3. First we declare R^Γ to be a , -congruence that extends the relation R, where R relates only `empty` and `empty'`:

```
th PROOF is pr STACK2 .
  op _R^Γ_ : Stack Stack -> Bool [comm] .
  var S S' S'' : Stack .
  var N : Nat .
  eq  S R^Γ S  =  true .
  cq  S R^Γ S''  =  true  if  S R^Γ S' and S' R^Γ S'' .
  cq  push(N,S) R^Γ push(N,S')  =  true  if  S R^Γ S' .
  eq  empty R^Γ empty'  =  true .
endth
```

Note that we do not explicitly assume that R^Γ extends $\equiv_\Sigma$, since behavioral equivalence is denoted by equality in the specification `STACK2`. This means that a proof using equational deduction will be sound only when R^Γ extends $\equiv_\Sigma$, which is why that requirement is included in the conditions to Proposition 61. For example, we can show that condition (1) of Proposition 61 is met with the following reductions:

```
red pop empty R^Γ pop empty' .
red top empty == top empty' .
```

The first reduction uses the equation `pop empty = empty`, which need be only behaviorally satisfied by a `STACK2`-model; the assumption that R^Γ extends behavioral Σ-equivalence means that these reductions really do prove that condition (1) is satisfied. To complete the proof, we show condition (2) of Proposition 61:

```
open PROOF .
ops s s' : -> Stack .
op n : -> Nat .
eq s R^Γ s' = true .
red pop push(n,s) R^Γ pop push(n,s') .
red top push(n,s) == top push(n,s') .
close
```

The theory `PROOF` says that R^Γ is *some* , -congruence that extends R. But the reductions are valid for any model of `PROOF`, so they are valid for all models that interpret R^Γ as the least such congruence, which is what we need. We use equational logic not to prove properties of all hidden `STACK2`-models, but rather to prove properties of congruences R^Γ. In this case, we have shown that all models satisfy the conditions of Proposition 61; the details of these proofs are 'hidden' by the use of OBJ reductions, but are very straightforward and can easily be reconstructed by the interested reader, or printed with OBJ3's trace facility. Note that the use of hidden constants to eliminate universal quantifiers over hidden variables relies on a "Theorem of Hidden Constants," as proved in [58].

5 Hidden Queries

Suppose we want to know if every object of some class (regardless of how that class is implemented) can be put into a state that satisfies certain constraints; for example, we might ask "can the elements of a certain

stack be put in increasing order?" In general, such queries could involve several objects. Our approach to the semantics of the object paradigm suggests that we formalize this situation by regarding the constraints as behavioral equations (more generally, Section 6 shows how to use first order predicates in constraints), and grouping them together in an *existential query* with an explicit declaration of the logical variables for which we seek values. A *solution* to the query will consist of values for the logical variables such that the equations are behaviorally satisfied by every hidden P-algebra.

To make this computationally feasible, we would like to find a term algebra that is "representative" for all other P-algebras, in the sense that a solution to a query in this algebra systematically translates to a solution in any other. Our "hidden Herbrand Theorem" says that this is possible in many interesting cases; in fact, we can use the initial P-algebra, just as in the ordinary Horn clause case [35]. By Theorem 46, this requires a consistent, lexic theory. However, we also show that even without these restrictions, equational deduction, and therefore techniques such as narrowing and paramodulation, are sound for arbitrary hidden theories. (Please recall that we start with hidden equational theories, but later extend to theories over hidden Horn clause logic with equality.)

Definition 63: Given a hidden signature Σ, an (**existential**) **Σ-query** is a sentence q of the form

$$(\exists X)\, t_1 = t'_1, \ldots, t_m = t'_m$$

where $t_j, t'_j \in T_\Sigma(X)_{s_j}$ for $j = 1, ..., m$. A hidden Σ-algebra A **behaviorally satisfies** q, written $A \models_\Sigma q$, iff there is an assignment $\theta\colon X \to A$ such that $\theta^*(c_j[t_j]) = \theta^*(c_j[t'_j])$ for all appropriate contexts c_j, for $j = 1, ..., m$. Call such an assignment a **solution** (or **witness**) for the query. □

Note that X can contain both visible and hidden variables. From the definition of behavioral equivalence we have the following:

Fact 64: Given an existential query q of the form $(\exists X)\, t_1 = t'_1, \ldots, t_m = t'_m$ and a Σ-algebra A, then A behaviorally satisfies q with solution $\theta\colon X \to A$ iff $\theta^*(t_j) \equiv_\Sigma \theta^*(t'_j)$ for $j = 1, ..., m$. □

Lemma 65: Given a Σ-query and a Σ-algebra A, if A satisfies the query, then A behaviorally satisfies it.

Proof: If A satisfies the query $(\exists X)\, t_1 = t'_1, \ldots, t_m = t'_m$, then there is some $\theta\colon X \to A$ such that $\theta^*(t_j) = \theta^*(t'_j)$ for $j = 1, ..., m$. This implies that $\theta^*(t_j) \equiv_A \theta^*(t'_j)$, so that θ also behaviorally solves the query, by Fact 64. □

Example 66: In the setting of Example 13, the behavioral query

$$(\exists S, S')\ \mathtt{push}(3, S) = \mathtt{pop}(S')$$

asks whether there are two stacks that are related in the indicated way, for any possible way of implementing `STACK`. One solution to this query is

$$\begin{aligned} S &= \mathtt{empty} \\ S' &= \mathtt{push}(0, \mathtt{push}(3, \mathtt{empty})) \end{aligned}$$

and of course there are also many others. □

The solution for this example can be found in the initial term algebra using narrowing, as in the language Eqlog [34, 11]. Then the unique homomorphism from it to any other algebra which satisfies `STACK` gives corresponding values in each of these algebras. However, it is not obvious that this technique can guarantee the *behavioral* satisfaction of the query in all algebras which behaviorally satisfy `STACK`. The results given below show that techniques such as term rewriting, narrowing and coinduction can indeed solve queries over all (Σ, E)-algebras.

Lemma 67: Given a hidden homomorphism $h\colon A \to B$ and a query q, if $A \models q$ then $B \models q$.

Proof: Suppose q is of the form $(\exists X)\, t_1 = t'_1, \ldots, t_m = t'_m$, and $A \models q$ with solution $\theta\colon X \to A$. Then by Lemma 22, $h \circ \theta\colon X \to B$ is a solution in B. □

Theorem 68: Given a hidden theory P:

(1) an initial P-algebra behaviorally satisfies a query iff all P-algebras behaviorally satisfy it; and

(2) a final P-algebra behaviorally satisfies a query iff some P-algebra behaviorally satisfies it.

Proof: Immediate from Lemma 67. □

Goguen and Meseguer [35, 34] gave a Herbrand theorem for Horn clause logic with equality, which states that an existential query is satisfied by the initial model of a specification iff it is satisfied by all models of that specification. Theorems 69 and 72 below give a Herbrand Theorem for hidden algebra, and a proof that techniques based on equational deduction are sound for arbitrary hidden theories. Section 6 generalizes this to hidden Horn clause logic with equality.

Theorem 69: (Hidden Herbrand Theorem) Given a consistent, lexic hidden theory P and a Σ-query q, then $I_P \models\!\!\models q$ iff every P-algebra behaviorally satisfies q.

Proof: I_P exists by Theorem 46, and the result now follows directly from Theorem 68. □

A weaker, but still useful corollary is

Proposition 70: Given a consistent, lexic hidden theory P and a query q, if I_P satisfies q then all P-algebras behaviorally satisfy q.

Proof: If I_P satisfies q then it behaviorally satisfies q by Lemma 65, so all P-algebras behaviorally satisfy q by Theorem 69. □

Corresponding to Proposition 59 we have the following, which justifies equational techniques in finding solutions to queries:

Theorem 71: Let q be a Σ-query of the form $(\exists X)\ t_1 = t'_1, \ldots, t_m = t'_m$. For a set E of Σ-equations, and an assignment $\theta : X \to T_\Sigma$, if $E \models (\forall \emptyset)\ \theta^*(c[t_i]) = \theta^*(c[t'_i])$ for all appropriate c and for $i = 1, \ldots, m$, then q is behaviorally satisfied by every hidden (Σ, E)-algebra.

Proof: If A is a hidden (Σ, E)-algebra, then $A/_{\equiv_\Sigma} \models E$, so $A/_{\equiv_\Sigma} \models (\forall \emptyset)\ \theta^*(c[t_i]) = \theta^*(c[t'_i])$ for each i and appropriate c, which means that $!_A(\theta^*(t_i)) \equiv_\Sigma !_A(\theta^*(t'_i))$, and so $A \models\!\!\models (\forall \emptyset)\ \theta^*(t_i) = \theta^*(t'_i)$ as desired. □

There is also a weaker form without contexts, corresponding to Proposition 58:

Theorem 72: Let q be a Σ-query of the form $(\exists X)\ t_1 = t'_1, \ldots, t_m = t'_m$. For a set E of Σ-equations, and an assignment $\theta : X \to T_\Sigma$, if $E \models (\forall \emptyset)\ \theta^*(t_i) = \theta^*(t'_i)$ for $i = 1, \ldots, m$, then q is behaviorally satisfied by every hidden (Σ, E)-algebra. □

The following illustrates the use of narrowing (e.g., see [9]) to solve queries. Narrowing can help discover if a query is satisfied by a term algebra, but it does not directly tell about behavioral satisfaction.

Example 73: We again use the `STACK2` specification in Section 4.2. Consider the query

```
(∃ S : Stack) top S = 3.
```

Using Theorem 72, we can obtain a solution by narrowing each side of the equation until we reach an equation $t = t'$ such that t unifies with t'; composing all the substitutions needed during this process gives us a solution. Thus, `top S` unifies with the left hand side of the equation `top push(N,S') = N` (with variables suitably renamed) with the substitution `S` $\mapsto$ `push(N,S')`, so that `top S` narrows to `N` which cannot be narrowed further. We then unify `N` with `3`, and to obtain a ground solution, we unify `S'` with `empty`. In this way obtain the solution

```
S = push(3,empty) .
```

(There are many other solutions.) This solution behaviorally satisfies the query for all STACK2-algebras. Moreover, from the proof of Theorem 72, it is clear that the expression given above provides a solution in any model.

Now consider the query:

```
(∃ S : Stack) pop S = empty, pop S = empty'.
```

Again, using narrowing we can try to find a solution for the first equation then apply the substitutions thus obtained to the second equation. Next, we look for a solution to this instance of the second equation. If this latter step fails then we must find another solution for the first equation and repeat the process.

We note that `pop S` unifies with the left hand side of the equation `pop empty = empty`, and so narrows to `empty`. This immediately gives us a solution `S = empty` to the first equation. Applying this substitution to `pop S = empty'` yields the equation `pop empty = empty'`. Reducing the left hand side, we end up with the goal `empty = empty'` which, as shown in Section 4.2, can be proved by coinduction. Therefore, we get the ground solution `S = empty`. Note that for this query, we rely on Theorem 71 to show that this is a solution for all hidden STACK2-algebras. □

6 Hidden Horn Clause Logic

The queries considered so far have all used equations as constraints. In relational programming languages such as Prolog and Eqlog [34], the sentences are Horn clauses with predicate symbols, which are interpreted as relations in models. This section shows that by generalizing a theorem of Diaconescu [10], we can lift our previous results to hidden Horn clause logic with equality.

Recall (e.g., from [22]) that a (**many sorted**) **first order signature** (**with equality**) is a triple (S, Σ, Π) such that (S, Σ) is a many sorted signature and Π is an S^+-indexed family of sets of **predicate** or **relation** symbols. We shall often write (Σ, Π) for (S, Σ, Π), leaving the sort set implicit. For every sort $s \in S$, there is a distinguished **equality** symbol $= \in \Pi_{s\,s}$. A **morphism** $(f, g, k) \colon (S, \Sigma, \Pi) \to (S', \Sigma', \Pi')$ between two first order signatures consists of a signature morphism (f, g) together with an S^+-indexed family of maps $k_w \colon \Pi_w \to \Pi'_{f^+(w)}$ on predicate symbols, where f^+ is f^* restricted to non-empty strings. A **model** M of a first order signature (S, Σ, Π) is a Σ-algebra together with an interpretation $M_\pi \subseteq M_w$ for each predicate symbol $\pi \in \Pi_w$, with the equality symbol always interpreted as true identity. A **morphism** $h \colon M \to M'$ between (S, Σ, Π)-models M and M' is a Σ-homomorphism such that for any predicate symbol $\pi \in \Pi_{s_1 \ldots s_n}$, if $(m_1, \ldots, m_n) \in M_\pi$, then $(h_{s_1}(m_1), \ldots, h_{s_n}(m_n)) \in M'_\pi$.

For a first order signature (Σ, Π), let $\mathbf{Mod}_{\Sigma,\Pi}$ denote the category of (Σ, Π)-models and morphisms. If $(S', \Sigma', \Pi') \subseteq (S, \Sigma, \Pi)$ is an inclusion of first order signatures, then there is a forgetful (reduct) functor $\restriction_{\Sigma',\Pi'} : \mathbf{Mod}_{\Sigma,\Pi} \to \mathbf{Mod}_{\Sigma',\Pi'}$ which maps any (Σ, Π)-model M to the (Σ', Π')-model $M\restriction_{\Sigma',\Pi'}$ whose carriers are the V-sorted carriers of M, with operations the Σ'-operations on M, and relations the Π'-relations on M. If $h \colon M \to M'$ is a morphism of (Σ, Π)-models, then $h\restriction_{\Sigma',\Pi'}$ is the (Σ', Π')-morphism obtained by restricting h to the sorts in Σ'.

Given a many sorted first order signature (Σ, Π) and an S-indexed set X of variables with the sets X_s disjoint, we can build the (Σ, Π)-**term model** $T_{\Sigma,\Pi}(X)$ with carriers and operations those of $T_\Sigma(X)$, and with $T_{\Sigma,\Pi}(X)_\pi = \emptyset$ for every $\pi \in \Pi$ except when π is the distinguished equality predicate symbol, which is interpreted as equality in $T_\Sigma(X)$. An **assignment** $\theta \colon X \to M$ is an S-sorted mapping from an S-indexed set of variables X to a (Σ, Π)-model M; it extends uniquely to a morphism $\theta^* \colon T_{\Sigma,\Pi}(X) \to M$. For a ground Σ-term t, we let M_t denote the element $!_M(t)$ of M, where $!_M \colon T_{\Sigma,\Pi}(\emptyset) \to M$ is the extension to a morphism of the unique assignment $\emptyset \to M$.

Definition 74: Given a first order signature (Σ, Π), a $\Pi(X)$-**atom** is a term of the form $\pi(t_1, \ldots, t_n)$, for π a predicate symbol in $\Pi_{s_1 \ldots s_n}$ and $t_i \in T_\Sigma(X)_{s_i}$. Given a (Σ, Π)-model M, an assignment $\theta \colon X \to M$ satisfies an atom $B = \pi(t_1, \ldots, t_n)$ iff $(\theta^*(t_1), \ldots, \theta^*(t_n)) \in M_\pi$, in which case we write $\theta \models_X B$. □

Definition 75: A (Σ, Π)-**Horn clause** is an expression of the form

$$(\forall X)\, B \;\; \textbf{if} \;\; B_1, \ldots, B_m$$

where $B, B_1, \ldots, B_m$ are all $\Pi(X)$-atoms. A Horn clause of the above form is said to be **unconditional** iff $m = 0$, and in that case it is written $(\forall X)\, B$.

Given a (Σ, Π)-Horn clause e of the form $(\forall X)\, B$ **if** $B_1, \ldots, B_m$, and a (Σ, Π)-model M, we say that M **satisfies** e, written $M \models_{\Sigma,\Pi} e$, iff for every assignment $\theta\colon X \to M$, we have $\theta \models_X B$ whenever $\theta \models_X B_j$ for $j = 1, \ldots, m$.

For a set C of (Σ, Π)-Horn clauses, let $\mathbf{Mod}_{\Sigma,\Pi,C}$ denote the full subcategory of $\mathbf{Mod}_{\Sigma,\Pi}$ whose objects are all models which satisfy each clause in C. □

Definition 76: A (Σ, Π)-**query** is an expression of the form

$$(\exists X)\, B_1, \ldots, B_m$$

where the B_j are $\Pi(X)$-atoms. If q is such a query, then a (Σ, Π)-model M **satisfies** q, written $M \models_{\Sigma,\Pi} q$, iff there is an assignment $\theta\colon X \to M$ such that $\theta \models_X B_j$ for $j = 1, \ldots, m$. □

As with equational logic, these concepts have hidden counterparts. We fix a universe (V, Ψ, Υ, D) of data values, where (V, Ψ, Υ) is a first order signature and D is a (V, Ψ, Υ)-model.

Definition 77: A **hidden first order signature** (over (V, Ψ, Υ, D)) is a first order signature (Σ, Π) where Σ is a hidden (equational) signature, $\Upsilon \subseteq \Pi$, and

(S1′) $\Pi_w \subseteq \Upsilon_w$ for $w \in V^+$;

(S2′) if $\pi \in \Pi_w$ for $w \in S^+$, then w has at most one element in H.

□

For convenience, assume that for any $\pi \in \Pi_w$ with a hidden sorted argument, that argument is its first.

Definition 78: For a hidden first order signature (Σ, Π), a hidden (Σ, Π)-**model** is a (Σ, Π)-model M such that $M|_{\Psi,\Upsilon} = D$. A **morphism** $h\colon M \to M'$ between hidden (Σ, Π)-models is a morphism of many sorted models such that $h|_{\Psi,\Upsilon} = 1_D$. We let $\mathbf{HMod}^D_{\Sigma,\Pi,\Psi,\Upsilon}$ denote the category of all hidden (Σ, Π)-models and their morphisms. We may write $\mathbf{HMod}_{\Sigma,\Pi}$ when the other elements of the signature are clear from the context. □

The notion of behavioral equivalence extends to Horn clause logic by requiring that no atoms distinguish equivalent states:

Definition 79: Given a hidden first order signature (Σ, Π), a (Σ, Π)-model M and $m, m' \in M_s$ for some $s \in S$, then m and m' are **behaviorally equivalent**, written $m \equiv_{M,s} m'$, or just $m \equiv_M m'$, iff either $s \in V$ and $m = m'$, or $s \in H$ and

- $m \equiv_\Sigma m'$, as in hidden equational logic, and
- for every predicate symbol $\pi \in \Pi_{s_1 \ldots s_n}$ and appropriate hidden context c of sort s_1 (i.e., $c \in L_\Sigma[z_s]_{s_i}$), and for all $d_i \in D_{s_i}$, $i = 2, \ldots, n$, we have

$$(M_c(m), d_2, \ldots, d_n) \in M_\pi \quad \text{iff} \quad (M_c(m'), d_2, \ldots, d_n) \in M_\pi \ .$$

□

Satisfaction now extends to hidden models as follows:

Definition 80: Given a (Σ, Π)-model M, an assignment $\theta\colon X \to M$ **behaviorally satisfies** a $\Pi(X)$-atom B iff B is of the form $t = t'$ and $\theta^*(t) \equiv_M \theta^*(t')$, or B is of the form $\pi(t_1, \ldots, t_n)$, where π is not the equality symbol, and $\theta \models_X B$. In both cases we write $\theta \models\!\!\equiv_X B$.

Given a (Σ, Π)-Horn clause e of the form $(\forall X)\, B$ **if** $B_1, \ldots, B_m$, and a (Σ, Π)-model M, we say that M **behaviorally satisfies** e, written $M \models\!\!\equiv_{\Sigma,\Pi} e$, iff for every assignment $\theta\colon X \to M$, we have $\theta \models\!\!\equiv_X B$ whenever $\theta \models\!\!\equiv_X B_j$ for $j = 1, \ldots, m$.

Given a (Σ, Π)-query q of the form $(\exists X)\, B_1, \ldots, B_m$ we say that a (Σ, Π)-model M **behaviorally satisfies** q, written $M \models\!\!\equiv_{\Sigma,\Pi} q$, iff there is an assignment $\theta\colon X \to M$ such that $\theta \models\!\!\equiv_X B_j$ for $j = 1, \ldots, m$. □

Diaconescu [10] gives a way of translating a first order signature into an algebraic signature by treating the predicate symbols as function symbols with result sort *Bool*, where *Bool* is a new sort with a single new constant *true*. Here we extend his definition to a translation between hidden signatures.

Definition 81: For any hidden first order signature (Σ, Π), define a hidden algebraic signature $(\Sigma \cup \Pi^b)$ over a data universe $(V^b, \Psi \cup \Upsilon^b, D^b)$ by

- $V^b = V \cup \{Bool\}$, where *Bool* is a new sort name;
- Π^b is a signature defined by $\Pi^b_{w,Bool} = \Pi_w$ and $\Pi^b_{w,s} = \emptyset$ for s different from *Bool*;
- Υ^b is a signature defined by $\Upsilon^b_{w,Bool} = \Upsilon_w$
- D^b is the $(\Psi \cup \Upsilon^b)$-algebra with $D^b_v = D_v$ for $v \in V$ and $D_{Bool} = \{true, false\}$, with Ψ-operation symbols interpreted as in D, and with $D^b_\pi(d_1, \ldots, d_n) = true$ if $(d_1, \ldots, d_n) \in D_\pi$ and *false* otherwise, for $\pi \in \Pi_{s_1 \ldots s_n}$ and $d_i \in D_{s_i}$.

□

Again generalizing Diaconescu [10], the corresponding translation from algebras to models uses an adjunction:

Definition 82: Given a hidden first order signature (Σ, Π), define the forgetful functor

$$W_{\Sigma,\Pi} : \mathbf{HAlg}^{D^b}_{\Sigma \cup \Pi^b} \to \mathbf{HMod}^D_{\Sigma,\Pi}$$

to map a $(\Sigma \cup \Pi^b)$-algebra A to the (Σ, Π)-model whose S-indexed carriers are those of A, with operations those of A restricted to Σ, and with relations in Π defined by $(a_1, \ldots, a_n) \in W(A)_\pi$ iff $A_\pi(a_1, \ldots, a_n) = true$. If $f : A \to A'$ is a morphism in $\mathbf{HAlg}_{\Sigma \cup \Pi^b}$ then $W_{\Sigma,\Pi}(f) = f$. We may write W instead of $W_{\Sigma,\Pi}$ if the context permits. □

Fact 83: $W_{\Sigma,\Pi}$ is well-defined.

Proof: We have to check that applying $W_{\Sigma,\Pi}$ to a hidden sorted algebra really does give a hidden sorted model. If A is a $(\Sigma \cup \Pi^b)$-algebra then $W_{\Sigma,\Pi}(A)|_{\Psi,\Upsilon}$ is the (Ψ, Υ)-model whose carriers are the visible sorted carriers of A. By the definition of D^b, this is the model D.

The preservation of composition and identity and the homomorphic property of $W_{\Sigma,\Pi}(f)$, for any homomorphism f in $\mathbf{HAlg}_{\Sigma \cup \Pi^b}$, follow easily from the fact that $W_{\Sigma,\Pi}(f) = f$. Finally, for such a homomorphism $f : A \to A'$, if $(a_1, \ldots, a_n) \in W(A)_\pi$ then $A_\pi(a_1, \ldots, a_n) = true$, so $A'_\pi(f(a_1), \ldots, f(a_n)) = true$, i.e., $(f(a_1), \ldots, f(a_n)) \in W(A')_\pi$. □

Theorem 84: Given a hidden signature (Σ, Π), $W_{\Sigma,\Pi}$ has an inverse $F_{\Sigma,\Pi}$.

Proof: For brevity, we write F instead of $F_{\Sigma,\Pi}$. Given a (Σ, Π)-model M the carriers of $F(M)$ are the same as those of M. The Σ-operation symbols are interpreted as in M and, for each $\pi \in \Pi_{s_1 \ldots s_n}$ with $m_i \in M_{s_i}$, $F(M)_\pi(m_1, \ldots, m_n) = true$ if $(m_1, \ldots, m_n) \in M_\pi$ and *false* otherwise. To show that $F(M)$ is indeed a hidden algebra, note that $F(M)|_{\Psi \cup \Upsilon^b}$ is the $(\Psi \cup \Upsilon^b)$-algebra whose carriers are the visible sorted carriers of M; by definition, this is the algebra D^b.

To see that W and F are each other's inverse, please note: that neither changes the carriers or Σ-interpretations of their arguments; that they alternately view symbols in Π as predicate symbols and as Boolean-valued operation symbols; and that for any (Σ, Π)-model M, predicate symbol $\pi \in \Pi_w$, and $m \in M_w$,

$$m \in W(F(M))_\pi \text{ iff } F(M)_\pi(m) = true \text{ iff } m \in M_\pi \ .$$

□

As in the above proof, we often write $F_{\Sigma,\Pi}$ as just F.

We now consider Horn clause specifications and their models.

Definition 85: Given a set C of (Σ, Π)-Horn clauses, let $\mathbf{HMod}^D_{\Sigma,\Pi,\Psi,\Upsilon,C}$ denote the full subcategory of $\mathbf{HMod}^D_{\Sigma,\Pi,\Psi,\Upsilon}$ whose objects behaviorally satisfy each clause in C. We shall often write this as $\mathbf{HMod}_{\Sigma,\Pi,C}$ and call its objects $(\Sigma, \Pi, \Psi, \Upsilon, C)$-**models** or just (Σ, Π, C)-**models** if context permits. □

Diaconescu [10] defines a translation from Horn clauses to conditional equations; we extend this to include queries:

Definition 86: Given a (Σ, Π)-Horn clause e and a (Σ, Π)-query q, define $\alpha(e)$, a conditional $(\Sigma \cup \Pi^b)$-equation, and $\alpha(q)$, a $\Sigma \cup \Pi^b$-query, as follows:

1. Every equation $t_1 = t_2$ is left untouched;
2. every atom $\pi(t_1, \ldots t_n)$ not of the above form is translated as $\pi(t_1, \ldots, t_n) = \mathit{true}$.

Also α extends in the obvious way to sets of Horn clauses. □

Lemma 87: Given a $(\Sigma \cup \Pi^b)$-algebra A and elements $a, a' \in A_s$ for some $s \in S$, we have

$$a \equiv_\Sigma a' \text{ iff } a \equiv_{W_{\Sigma,\Pi}(A)} a'.$$

Proof: The 'if' direction is straightforward, because for any Σ-context c, we have $W(A)_c = A_c$. Moreover, any $(\Sigma \cup \Pi^b)$-context is either a Σ-context or of the form $\pi(c[z], d_1, \ldots, d_n)$, in which case, if $a \equiv_{W_{\Sigma,\Pi}(A)} a'$, then a and a' give the same results in such contexts.

Conversely, if $a \equiv_\Sigma a'$ then for any $\pi \in \Pi_{h\, v_1 \ldots v_n}$ and appropriate hidden context $c \in L_\Sigma[z]_h$

$$(W(A)_c(a), d_1, \ldots, d_n) \in W(A)_\pi \quad \text{iff} \quad A_\pi(A_c(a), d_1, \ldots, d_n) = \mathit{true}$$

iff, because $a \equiv_\Sigma a'$,

$$A_\pi(A_c(a'), d_1, \ldots, d_n) = \mathit{true} \quad \text{iff} \quad (W(A)_c(a'), d_1, \ldots, d_n) \in W(A)_\pi,$$

and so $a \equiv_{W_{\Sigma,\Pi}(A)} a'$. □

Corollary 88: Given a $(\Sigma \cup \Pi^b)$-algebra A, for any $\Pi(X)$-atom B and any $\theta : X \to A$, we have $\theta \models\!\!\equiv_X B$ iff $\theta^*(t) \equiv_\Sigma \theta^*(t')$, where $\alpha(B) = (t = t')$.

Proof: Lemma 87 proves this for the case where B is the atom $(t = t')$. If B is of the form $\pi(t_1, \ldots, t_n)$, then $\theta \models\!\!\equiv_X B$ iff $(\theta^*(t_1), \ldots, \theta^*(t_n)) \in W(A)_\pi$, iff $A_\pi(\theta^*(t_1), \ldots, \theta^*(t_n)) = \mathit{true}$, iff $\theta^*(\pi(t_1, \ldots, t_n)) \equiv_\Sigma \theta^*(\mathit{true})$ as desired. □

Finally, we can formalize the validity of translating behavioral satisfaction of Horn clauses into behavioral satisfaction of conditional equations:

Proposition 89: For any $(\Sigma \cup \Pi^b)$-algebra A, we have

(1) for e a (Σ, Π)-Horn clause,

$$A \models^{D^b}_{\Sigma \cup \Pi^b} \alpha(e) \text{ iff } W_{\Sigma,\Pi}(A) \models^{D}_{\Sigma,\Pi} e$$

(2) for q a (Σ, Π)-query,

$$A \models^{D^b}_{\Sigma \cup \Pi^b} \alpha(q) \text{ iff } W_{\Sigma,\Pi}(A) \models^{D}_{\Sigma,\Pi} q$$

Proof: To see (1), let e be a Horn clause of the form $(\forall X)\, B$ **if** $B_1, \ldots, B_m$, and let $\alpha(e)$ be $(\forall X)\, t = t'$ **if** $t_1 = t'_1, \ldots, t_m = t'_m$. Then $A \models\!\!\equiv e$ is equivalent to: for every $\theta : X \to A$ we have $\theta^*(t) \equiv_\Sigma \theta^*(t')$ whenever $\theta^*(t_j) \equiv_\Sigma \theta^*(t'_j)$ for $j = 1, \ldots, m$. By Corollary 88, this is equivalent to saying that for every $\theta : X \to A$ we have $\theta \models\!\!\equiv_X B$ whenever $\theta \models\!\!\equiv_X B_j$ for $j = 1, \ldots, m$, which is equivalent to $W(A) \models\!\!\equiv e$. The proof of (2) is similar. □

Corollary 90: (Translation) Given a (Σ, Π)-Horn clause e and a (Σ, Π)-query q, if M is any (Σ, Π)-model then

(1) M behaviorally satisfies e iff $F_{\Sigma,\Pi}(M)$ behaviorally satisfies $\alpha(e)$, and

(2) M behaviorally satisfies q iff $F_{\Sigma,\Pi}(M)$ behaviorally satisfies $\alpha(q)$.

Proof: Since these assertions have very similar proofs, we only prove (1). By Proposition 89 and the fact that $F_{\Sigma,\Pi}$ is a left inverse for $W_{\Sigma,\Pi}$, we have

$$M \models^D e \text{ iff } W_{\Sigma,\Pi}(F_{\Sigma,\Pi}(M)) \models\!\!\models e \text{ iff } F_{\Sigma,\Pi}(M) \models\!\!\models \alpha(e) \ .$$

□

Proposition 89 and the above corollary imply that we can check whether a model behaviorally satisfies a Horn clause by testing whether an algebra behaviorally satisfies a conditional equation. We now examine initial and final models of Horn clause specifications.

Definition 91: A set C of (Σ,Π)-Horn clauses is **lexic** iff $\alpha(C)$ is lexic. □

Using this, we can extend Theorem 69 to obtain a hidden Herbrand theorem for hidden sorted Horn clause logic with equality:

Theorem 92: (Hidden Herbrand Theorem) Given a consistent lexic set C of (Σ,Π)-Horn clauses, then $I_P \models\!\!\models \alpha(q)$ iff $M \models\!\!\models q$ for every model M in $\mathbf{HMod}_{\Sigma,\Pi,C}$, where $P = (\Sigma \cup \Pi^b, \alpha(C))$.

Proof: By Theorem 69, $I_P \models\!\!\models \alpha(q)$ iff every P-algebra behaviorally satisfies $\alpha(q)$; by Corollary 90, this in turn is equivalent to saying that q is behaviorally satisfied by every C-model M. □

In fact, all of the results of the previous section can be pushed through the equivalence between hidden equational algebras and hidden first order models, in the same way as the above Herbrand theorem. For example, narrowing for first order specifications is justified by Theorem 71 to give

Theorem 93: Let q be a (Σ,Π)-query such that $\alpha(q)$ is of the form $(\exists X)\ t_1 = t'_1, \ldots, t_m = t'_m$. For a set C of (Σ,Π)-Horn clauses, and assignment $\theta : X \to T_\Sigma$, if $\alpha(C) \models (\forall \emptyset)\ \theta^*(c[t_i]) = \theta^*(c[t'_i])$ for all appropriate c and for $i = 1, \ldots, m$, then q is behaviorally satisfied by every hidden (Σ,Π,C)-model. □

Example 94: To illustrate this result, we add a relation, `ascending`, to the `STACK` theory of Example 13, defined by two axioms, using a notation like that of Eqlog [34, 11],

```
rel ascending : Stack .
var N : Nat .
var S : Stack .
ax ascending(empty) .
ax ascending(push(N,S)) if N > top(S), ascending(S) .
```

where `_>_ : Nat Nat` is a built in predicate symbol. To make things more interesting, let us also add a new attribute `height`, defined as follows,

```
op height : Stack -> Nat .
var N : Nat .
var S : Stack .
eq height(empty) = 0 .
eq height(push(N,S)) = 1 + height(S) .
```

giving a Horn clause specification `STACKA`. Translating to a strictly equational form would mean replacing the definition of `ascending` with

```
op ascending : Stack -> Bool .
var N : Nat .
var S : Stack .
eq ascending(empty) = true .
eq ascending(push(N,S)) = true  if N > top(S) = true, ascending(S) = true .
```

to give us an equational specification STACKE (the definition of height is already equational and so remains the same). We assume that the sort Bool is a visible sort with constants true and false; note that we have translated the predicate _>_ into a function symbol _>_ : Nat Nat -> Bool.

Now suppose we want to know whether

```
(∃ S : Stack) ascending(S), height(S) = 4
```

is behaviorally satisfied by all models of STACKM. By Theorem 93, we can try to do this using equational techniques on the specification STACKE. For example, narrowing reduces the goal ascending(S) = true to

```
N > top(S') = true, ascending(S') = true
```

with S = push(N,S'); thus height(S) = 4 reduces to 1 + height(S') = 4. Iterating this by means of backtracking amongst the equations and narrowing, we reach a solution (among many others),

```
S = push(4,push(3,push(2,push(1,empty)))),
```

and Theorem 93 shows that this is a solution for all STACKA-models. □

7 Conclusions

This paper lays foundations for a novel programming paradigm combining the advantages of the logic, object, and functional paradigms. The Herbrand theorems are our main results, but the hidden model theory in Sections 3 and 4 further develops the research programme of [31], and the coinductive proof technique of Section 4 is useful in hidden algebra and related coalgebraic approaches [56, 42, 46, 8]. The consistency results in Section 3.3.1 are novel and useful.

The hidden approach differs from classical algebraic approaches in using behavioral satisfaction except for a fixed interpretation of visible sorts. This loose semantics allows hidden algebra to capture nondeterminism by underspecification [31]. Hidden algebra differs from Diaconescu's categorical approach to the constraint paradigm [11, 12], which has loose ordinary satisfaction even for its built in data types.

We have shown that a hidden theory has initial and final models iff it is consistent and lexic. The final model consists of abstract behaviors, and an equation is behaviorally satisfied by all models iff it is behaviorally satisfied by the final model. Dually, the initial, term-based model behaviorally satisfies an existential query iff all models behaviorally satisfy it. This gives rise to the two Herbrand theorems, for hidden equational logic and hidden Horn clause logic, which allow solutions to be constructed in initial term algebras. There is no completeness result for hidden algebra [5]; intuitively, solving constraints in hidden specifications can be arbitrarily complex; however, coinduction techniques can considerably simplify proofs, and often (e.g., in the FLAG example) reduce behavioral satisfaction to standard satisfaction. Such techniques have been implemented in the CafeOBJ [14, 13] algebraic specification language, and the more recent BOBJ [25] language and system implements some even more advanced techniques.

A useful direction for future research is to extend our results to include the kind of subtyping given by order sorted algebra [37]. Burstall and Diaconescu [4] have extended the hiding process to many other institutions, and in particular, to order sorted algebra (in the sense of [37, 23]). Malcolm and Goguen [48] show that hidden order sorted logic forms an institution, using a construction that differs from Burstall and Diaconescu's in its treatment of error-handling; yet another treatment of ordered sorts in hidden algebra preserves the relationship between hidden algebra and coalgebra [6]. The relationships between these different extensions of hidden algebra need further study.

References

[1] Krzystof Apt. *From Logic Programming to Prolog.* Prentice-Hall, 1997.

[2] Michael Barr and Charles Wells. *Category Theory for Computing Science.* Prentice-Hall, 1990.

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

[4] Rod Burstall and Răzvan Diaconescu. Hiding and behaviour: an institutional approach. In Andrew William Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*, pages 75–92. Prentice-Hall, 1994.

[5] Samuel Buss and Grigore Roşu. Incompleteness of behavioral logics. In Horst Reichel, editor, *Proceedings, Coalgebraic Methods in Computer Science (CMCS'00)*, volume 33 of *Electronic Notes in Theoretical Computer Science*, pages 61–79. Elsevier Science, March 2000.

[6] Corina Cîrstea, Grant Malcolm, and James Worrell. Hidden order sorted algebra: subtypes for objects, 1999. Draft, Department of Computer Science, University of Liverpool.

[7] Paul M. Cohn. *Universal Algebra*. Harper and Row, 1965. Revised edition 1980.

[8] Corina Cîrstea. Coalgebra semantics for hidden algebra: parameterized objects and inheritance. In Francisco Parisi-Presicce, editor, *12th Workshop on Algebraic Development Techniques*, pages 174–189. Springer, 1998. Lecture Notes in Computer Science, Volume 1376.

[9] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewriting systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Methods and Semantics*, pages 243–320. North-Holland, 1990.

[10] Răzvan Diaconescu. The logic of Horn clauses is equational. Technical Report PRG–TR–3–93, Programming Research Group, University of Oxford, 1993. Written 1990.

[11] Răzvan Diaconescu. *Category-based Semantics for Equational and Constraint Logic Programming*. PhD thesis, Programming Research Group, Oxford University, 1994.

[12] Răzvan Diaconescu. A category-based equational logic semantics to constraint programming. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *Recent Trends in Data Type Specification*, pages 200–222. Springer, 1996. Lecture Notes in Computer Science, Volume 389.

[13] 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.

[14] Răzvan Diaconescu and Kokichi Futatsugi. Logical semantics for CafeOBJ. In *Precise Semantics for Software Modelling Techniques*, pages 31–54. Technical University of München, 1998. Report TUM–19803, Proceedings of an ICSE'98 workshop (Kyoto, Japan).

[15] Răzvan Diaconescu and Kokichi Futatsugi. Behavioural coherence in object-oriented algebraic specification. *Journal of Universal Computer Science*, 6(1):74–96, 2000. Also technical report IS–RR–98–0017F, Japan Advanced Institute for Science and Technology, June 1998.

[16] Marie-Claude Gaudel and Igor Privara. Context induction: an exercise. Technical Report 687, LRI, Université de Paris-Sud, 1991.

[17] Joseph Goguen. Principles of parameterized programming. In Ted Biggerstaff and Alan Perlis, editors, *Software Reusability, Volume I: Concepts and Models*, pages 159–225. Addison Wesley, 1989.

[18] Joseph Goguen. Hyperprogramming: A formal approach to software environments. In *Proceedings, Symposium on Formal Approaches to Software Environment Technology*. Joint System Development Corporation, Tokyo, Japan, January 1990.

[19] 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.

[20] Joseph Goguen. An approach to situated adaptive software. In *Proceedings, International Workshop on New Models of Software Architecture*, pages 7–20. NEDO, 1993. (Kanazawa, Japan).

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

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

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

[24] Joseph Goguen and Răzvan Diaconescu. Towards an algebraic semantics for the object paradigm. In Hartmut Ehrig and Fernando Orejas, editors, *Proceedings, Tenth Workshop on Abstract Data Types*, pages 1–29. Springer, 1994. Lecture Notes in Computer Science, Volume 785.

[25] Joseph Goguen, Kai Lin, and Grigore Roşu. Circular coinductive rewriting. In *Proceedings, Automated Software Engineering '00*, pages 123–131. IEEE, 2000. (Grenoble, France).

[26] Joseph Goguen and Grant Malcolm. Proof of correctness of object representation. In Andrew William Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*, pages 119–142. Prentice-Hall, 1994.

[27] Joseph Goguen and Grant Malcolm. Situated adaptive software: beyond the object paradigm. In *Proceedings, International Symposium on New Models of Software Architecture*, pages 126–142. Information-Technology Promotion Agency, 1995. (Tokyo, Japan).

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

[29] Joseph Goguen and Grant Malcolm. Hidden coinduction: Behavioral correctness proofs for objects. *Mathematical Structures in Computer Science*, 9(3):287–319, June 1999.

[30] Joseph Goguen and Grant Malcolm, editors. *Software Engineering with OBJ: Algebraic Specification in Action*. Kluwer, 2000.

[31] Joseph Goguen and Grant Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, August 2000. Also UCSD Dept. Computer Science & Eng. Technical Report CS97–538, May 1997.

[32] Joseph Goguen, Grant Malcolm, and Tom Kemp. A hidden Herbrand theorem: Combining the object, logic and functional paradigms. In Catuscia Palamidessi, Hugh Glaser, and Karl Meinke, editors, *Principles of Declarative Programming*, pages 445–462. Springer Lecture Notes in Computer Science, Volume 1490, 1998.

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

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

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

[36] Joseph Goguen and José Meseguer. Unifying functional, object-oriented and relational programming, with logical semantics. In Bruce Shriver and Peter Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 417–477. MIT, 1987.

[37] Joseph Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992. Drafts exist from as early as 1985.

[38] Joseph Goguen and Grigore Roşu. Hiding more of hidden algebra. In Jeannette Wing, Jim Woodcock, and Jim Davies, editors, *FM'99 – Formal Methods*, pages 1704–1719. Springer, 1999. Lecture Notes in Computer Sciences, Volume 1709, Proceedings of World Congress on Formal Methods, Toulouse, France.

[39] 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.

[40] Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ. In Joseph Goguen and Grant Malcolm, editors, *Software Engineering with OBJ: Algebraic Specification in Action*, pages 3–167. Kluwer, 2000. Also Technical Report SRI-CSL-88-9, August 1988, SRI International.

[41] Jacques Herbrand. Recherches sur la théorie de la démonstration. *Travaux de la Société des Sciences et des Lettres de Varsovie, Classe III*, 33(128), 1930.

[42] Bart Jacobs. Inheritance and cofree constructions. In Pierre Cointe, editor, *European Conference on Object-Oriented Programming*, pages 210–231. Springer, 1996. Lecture Notes in Computer Science, Volume 1098.

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

[44] John Wilcox Lloyd. *Foundations of Logic Programming*. Springer, 1987. Second edition.

[45] Anatoly Ivanovich Malcev. *Algebraic Systems*. Springer, 1973.

[46] Grant Malcolm. Behavioural equivalence, bisimilarity, and minimal realisation. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *Recent Trends in Data Type Specifications: 11th Workshop on Specification of Abstract Data Types*, pages 359–378. Springer Lecture Notes in Computer Science, Volume 1130, 1996. (Oslo Norway, September 1995).

[47] Grant Malcolm. Interconnection of object specifications. In Stephen Goldsack and Stuart Kent, editors, *Formal Methods and Object Technology*, pages 359–378. Springer, 1996. Workshops in Computing.

[48] Grant Malcolm and Joseph Goguen. Proving correctness of refinement and implementation. Technical Report Technical Monograph PRG–114, Programming Research Group, University of Oxford, 1994.

[49] Karl Meinke and John V. Tucker. Universal algebra. In *Handbook of Logic in Computer Science, Volume 1*. Oxford, 1993.

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

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

[52] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, Parts I and II. Technical Report ECS-LFCS-89-85 and -86, Computer Science Department, University of Edinburgh, 1989.

[53] Robin Milner and Mads Tofte. Co-induction in relational semantics. *Theoretical Computer Science*, 87(1):209–220, 1991.

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

[55] Horst Reichel. Behavioural validity of conditional equations in abstract data types. In *Contributions to General Algebra 3*. Teubner, 1985. Proceedings of the Vienna Conference, June 21-24, 1984.

[56] Horst Reichel. An approach to object semantics based on terminal co-algebras. *Mathematical Structures in Computer Science*, 5:129–152, 1995.

[57] Grigore Roşu. Behavioral coinductive rewriting. In Kokichi Futatsugi, Joseph Goguen, and José Meseguer, editors, *OBJ/CafeOBJ/Maude at Formal Methods '99*, pages 179–196. Theta (Bucharest), 1999. Proceedings of a workshop in Toulouse, 20 and 22 September 1999.

[58] Grigore Roşu and Joseph Goguen. Hidden congruent deduction. In Ricardo Caferra and Gernot Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, pages 252–267. Springer, 2000. Lecture Notes in Artificial Intelligence, Volume 1761; papers from a conference held in Vienna, November 1998.

[59] Wolfgang Wechler. *Universal Algebra for Computer Scientists*. Springer, 1992. EATCS Monographs on Theoretical Computer Science, volume 25.