.40

Inductive Behavioral Proofs by Unhiding

Grigore Roşu

Department of Computer Science
University of Illinois at Urbana-Champaign, USA
http://cs.uiuc.edu/grosu

Abstract

We show that for any behavioral Σ-specification $\mathcal{B}$ there is an ordinary algebraic specification $\tilde{\mathcal{B}}$ over a larger signature, such that a model behaviorally satisfies $\mathcal{B}$ iff it satisfies, in the ordinary sense, the Σ-theorems of $\tilde{\mathcal{B}}$. The idea is to add machinery for contexts and experiments (sorts, operations and equations), use it, and then hide it. We develop a procedure, called *unhiding*, which takes a finite $\mathcal{B}$ and produces a finite $\tilde{\mathcal{B}}$. The practical aspect of this procedure is that one can use any standard equational inductive theorem prover to derive behavioral theorems, even if neither equational reasoning nor induction is sound for behavioral satisfaction.

1 Introduction

Information hiding is an important technique in modern programming. Programmers and software engineers agree that a crucial feature of the implementation languages they use, e.g. C++, Java, etc., is the support that these languages provide for both *public* and *private* entities (types, functions). The public part is often called *interface* and is visible to all the other modules (classes, packages), while the private one can only be internally used to implement the interface. Hiding implementation features allows not only an increased level of abstraction, but also an increased potential to improve a given data representation without having to search through all of a large program for each place where details of the representation are used. Parnas [35] discusses in depth the practical importance of hiding implementation details.

Information hiding is important not only in software development and modern programming, but also in algebraic specification. Majster [31] suggested that algebraic specifications are practically limited because certain Σ-algebras cannot be specified as an initial Σ-algebra of a finite set of Σ-equations, but later, Bergstra and Tucker [2] (see also [32]) showed that in fact any computable Σ-algebra can be specified as the Σ-restriction of an initial Σ'-algebra of a finite set of Σ'-equations, for some finite Σ' larger than Σ.

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

Therefore, there are some Σ-theories of interest that do not admit finite Σ-specifications but are Σ-restrictions of finitely presented Σ'-theories for some $\Sigma \subseteq \Sigma'$. Diaconescu, Goguen, Stefaneas [15] present logic paradigm independent (or *institutional* [18]) approaches to information hiding and integration of it with other operations on modules. Work on module algebra by Bergstra, Heering and Klint [1] also investigates information hiding formally.

Behavioral abstraction is another development in algebraic specification which appears under various names in the literature such as *hidden algebra* in works by Goguen, Diaconescu and many others [17,19,23,22,40,25], *observational logic* in works by Hennicker, Bidoit and many others [28,8,4,3], *coherent hidden algebra* in Diaconescu [14], *hidden logic* in Roşu [38], and so on. Most of these approaches appeared as a need to extend algebraic specifications to ease the process of specifying and verifying designs of systems and also for various other reasons, such as, to naturally handle infinite types [1], to give semantics to the object paradigm, to specify finitely otherwise infinitely axiomatizable abstract data types, etc. The main characteristic of these approaches is that sorts are split into *visible* (or *observational*) for data and *hidden* for states, and the equality is behavioral, in the sense that two states are *behaviorally equivalent* if and only if they *appear* to be the same under any visible *experiment*. The intuitions for behavioral abstraction go back at least to Montanari 1976 [16], Reichel 1981 [36,37], Goguen and Meseguer 1982 [24], and to Sannella and Tarlecki 1987 [41]. A closely related and elegant subject is *coalgebra* (for example see Jacobs and Rutten [29]): in many situations of interest, but certainly not in all interesting ones, *bisimulation* becomes a special case of behavioral equivalence, the coalgebraic *coinduction* proof principle extends to general behavioral specifications and, together with behavioral rewriting, it yields a powerful proof technique for behavioral equivalence [20,21,22,40]. We suggest [38], which is publicly available on the web, for an extensive study of behavioral specification in the hidden algebraic style, including algorithms for automated behavioral proving and references to related work. From now on in this paper we'll use the terminology of hidden logic as in [38], mentioning that the results apply to all approaches to behavioral abstraction based on behavioral equivalence that we are aware of. A *behavioral Σ-specification*, usually written $\mathcal{B}, \mathcal{B}', \mathcal{B}_1, ...$, is a triple (Σ, Γ, E) where Γ, the set of *behavioral operations*, is a subsignature of the S-sorted signature Σ and $S = V \cup H$ (V for visible and H for hidden sorts). The models of a behavioral specification are special algebras called *hidden algebras*.

The main theoretical goal of the present work is to relate the two important extensions of algebraic specification, namely information hiding and behavioral abstraction. We show that any equational behavioral Σ-specification is semantically equivalent to the Σ-restriction of an ordinary algebraic specification over a larger signature, thus emphasizing once more the definitional

[1] I.e., types whose values are infinite structures.

power of information hiding. More precisely, we show that for any behavioral Σ-specification $\mathcal{B} = (\Sigma, \Gamma, E)$ there is some specification $\tilde{\mathcal{B}} = (\Sigma', E')$ for some $\Sigma \subseteq \Sigma'$, called the *unhiding* of $\mathcal{B}$, such that a hidden Σ-algebra behaviorally satisfies $\mathcal{B}$ iff it strictly satisfies $\Sigma\Box\tilde{\mathcal{B}}$, which is the Σ-theory of all Σ-theorems of $\tilde{\mathcal{B}}$. Moreover, E' is finite whenever E is finite, and $\tilde{\mathcal{B}}$ can be generated automatically from $\mathcal{B}$. Even further, $\tilde{\mathcal{B}}$ is generated in such a way that inductive equational theorem provers can be used to prove behavioral equivalence in $\mathcal{B}$.

The general idea of unhiding in this paper is taken from [25], which was inspired from [6], but the technical constructions are radically changed. That is because we want to illustrate an important practical aspect of unhiding, namely its relationship to proving behavioral properties inductively, in particular to Hennicker's context induction proof principle [27]. Previous work by Bidoit, Hennicker [7] and Mikami [33] was also a great source of inspiration.

This paper contains algebraic definitions and proofs. We assume the reader familiar with general notions of algebra and many-sorted equational logics, such as initial algebra, morphism, satisfaction. If Σ is an S-sorted signature, $V \subseteq S$ and A is a Σ-algebra, then $\Sigma{\restriction_V}$ is the V-reduct of Σ and $A{\restriction_V}$ is the V-reduct of A, i.e., the V-sorted set obtained from A by forgetting its algebraic structure. If $\varphi\colon \Sigma \to \Sigma'$ is a signature morphism and A' is a Σ' algebra, then $A'{\restriction_\varphi}$ denotes the φ-reduct of A' to a Σ-algebra. If e is a Σ-equation then $\varphi(e)$ is its translation to a Σ'-equation. It is known as "satisfaction condition property" that in the context above, $A' \models_{\Sigma'} \varphi(e)$ iff $A'{\restriction_\varphi} \models_\Sigma e$.

We use Maude [12] equational notation in the two examples that we follow in the paper. We find it very intuitive so we don't describe it here, mentioning that it is almost identical to the OBJ notation [26].

2 Reachability and Induction

Induction is not sound for all the models of a specification, but only for the reachable ones, that are, those for which the unique morphism from the initial model is surjective. We need a more general approach to reachability and induction in this paper because of the special structure of our models.

Definition 2.1 Let (S, Σ) be a many-sorted signature. Given $S' \subseteq S$ and an $(S - S')$-indexed set Z of variables [2], a Σ-algebra A is (S', Z)**-reachable** iff for all $s' \in S'$ and $a' \in A_{s'}$ there is $\theta\colon Z \to A$ and $t' \in T_{\Sigma,s'}(Z)$ s.t. $\theta(t') = a'$.

The usual notion of reachability is a special case of the above when $S' = S$. The importance of (S', Z)-reachability is captured by the following

Proposition 2.2 *In the context of Definition 2.1, if A is (S', Z)-reachable and if $\mathcal{P}^T$ and $\mathcal{P}^A$ are S'-indexed predicates on $T_\Sigma(Z){\restriction_{S'}}$ and $A{\restriction_{S'}}$, respectively, s.t. for any $s' \in S'$ and $t' \in T_{\Sigma,s'}(Z)$ it is the case that $\mathcal{P}^T(t')$ implies $\mathcal{P}^A(\theta(t'))$ for any $\theta\colon Z \to A$, then $\mathcal{P}^T = T_\Sigma(Z){\restriction_{S'}}$ implies $\mathcal{P}^A = A{\restriction_{S'}}$.*

[2] By abuse of language, Z also denotes the S-indexed set with $Z_s = \emptyset$ for all $s \in S'$.

In practice, $\mathcal{P}^A$ is a property that one wants to show for all elements of sorts S' of a model A. The proposition above says that if A is (S', Z)-reachable then one can just find a "similar" property on the term model over variables in Z and prove the new property for all the terms of sorts in S'. This proof can be done by induction or by any other proof technique on term algebras.

3 Hidden Logics and Behavioral Abstraction

Hidden algebra extends algebraic specification to handle states in a natural way, using behavioral equivalence. Systems need only satisfy their requirements behaviorally, in the sense of *appearing* to satisfy them under all possible experiments. Hidden algebra was introduced in [17] and developed further in [19,9,23,39,40,14,11,22,25,30,20,21,38] among other places. Two systems, CafeOBJ [13] and BOBJ [20,21,38], supporting behavioral specification and reasoning have been implemented, both extending OBJ [26]. A comprehensive presentation of hidden algebra can be found in [38]. One distinctive feature of hidden algebra logics is to split sorts into *visible* for data and *hidden* for states. A model, or *hidden algebra*, is an abstract implementation, consisting of the possible states, with functions for operations. The restriction of a model to the visible subsignature is called *data*. *Hidden logics* [38] refer to close relatives of hidden algebra.

Definition 3.1 Given disjoint sets V, H called **visible** and **hidden sorts**, a **loose data hidden** (V, H)**-signature** is a many sorted $(V \cup H)$-signature. A **fixed data hidden** (V, H)**-signature** is a pair (Σ, D) where Σ is a loose data hidden (V, H)-signature and D, called the **data algebra**, is a many sorted $\Sigma\restriction_V$-algebra. A **loose data hidden subsignature of** Σ is a loose data hidden (V, H)-signature Γ with $\Gamma \subseteq \Sigma$ and $\Gamma\restriction_V = \Sigma\restriction_V$. A **fixed data hidden subsignature of** (Σ, D) is a fixed data hidden (V, H)-signature (Γ, D) over the same data with $\Gamma \subseteq \Sigma$ and $\Gamma\restriction_V = \Sigma\restriction_V$. The operations in Σ with one hidden argument and visible result are called **attributes**, those with one hidden argument and hidden result are called **methods**, those with two hidden arguments and hidden result are called **binary methods**, and those with only visible arguments and hidden result are called **hidden constants**.

Hereafter we may write "hidden signature" instead of "loose data hidden (V, H)-signature" or "fixed data hidden (V, H)-signature," and Σ for (Σ, D).

Example 3.2 Set. The hidden signature of sets of natural numbers is defined as follows: $\Sigma\restriction_V$ is the signature of natural numbers, including visible sorts *Nat* and *Bool*; H has one sort, *Set*; Σ adds the hidden constant $empty :\rightarrow Set$, the attribute $\_ \in \_ : Nat \times Set \rightarrow Bool$, the method $add : Nat \times Set \rightarrow Set$ for adding a new element to a set, and the binary methods $\_ \cup \_$, $\_ \cap \_ : Set \times Set \rightarrow Set$ for union and intersection, respectively. In the fixed-data approach, a fixed algebra of natural numbers is also considered.

Example 3.3 Stream. The hidden signature of infinite streams of numbers is as follows: $\Sigma\restriction_V$ is the signature of natural numbers providing a visible sort *Nat*; H has one sort, *Stream*; Σ adds an attribute $head : Stream \to Nat$ for the head of a stream, methods *tail*, *odd*, *even* : $Stream \to Stream$ for the tail stream, the streams of elements on odd and even positions, respectively, a method $\_\&\_ : Nat \times Stream \to Stream$ putting a number at the beginning of a stream, and a binary merging method $zip : Stream \times Stream \to Stream$.

Definition 3.4 A **loose data hidden Σ-algebra** A is a Σ-algebra, and a **fixed data hidden (Σ, D)-algebra** A is a Σ-algebra A such that $A\restriction_{\Sigma_V} = D$.

The first definition of hidden algebra was fixed-data [17], reason for which we call the other one loose-data. One may argue that one should only focus on loose-data hidden algebra and thus simplify all the remaining definitions in the paper. However, fixed-data hidden algebra has interesting theoretical and practical applications. For example, under certain monadicity restrictions with respect to the number of hidden arguments of operations, the category of fixed-data hidden algebras over a given fixed-data hidden signature is isomorphic to a category of coalgebras; on the other hand, a protocol like alternating bit protocol cannot be shown correct unless data is assumed distinct, in particular 0 different from 1. We therefore prefer to develop our results in a general setting that include both loose-data and fixed-data approaches.

Example 3.5 Set (Continued). A typical hidden algebra for sets of natural numbers has sets of numbers as elements of sort *Set*, and defines the operations as expected. However, another interesting model has *lists* of numbers as hidden elements, implements union as append and intersection by taking the list of those elements in the first list that occur in the second. This is how sets are implemented in LISP; note that multiple occurrences of elements are allowed.

Example 3.6 Stream. (Continued). The intended stream hidden algebra has infinite lists as hidden elements and defines the four operations in the obvious way. However, there also are less standard models, which, for example, view streams as infinite trees and implement the operations accordingly.

Unless specified otherwise, for the rest of the paper we fix a hidden signature Σ and a subsignature of it, Γ. A Σ-algebra should be regarded as a universe of possible states of a system. A system can be regarded as a "black-box," the inside of which is not seen, one being only concerned with its behavior under "experiments" with operations in Γ. Informally, an experiment is an observation of an attribute of a system after it has been perturbed, using the concept of context; the symbol $\bullet$ below is a placeholder for the state being experimented upon. The use of only a subset Γ of operators in Σ, often called *behavioral*, was a major decision in both behavioral specification and verification systems CafeOBJ and BOBJ, due not only to the natural desire to generate contexts using a reduced set of operators, but especially to the necessity of providing support for nondeterminism in these systems.

Definition 3.7 An (appropriate) Γ**-context for sort** s is a term in $T_\Gamma(\{\bullet : s\} \cup Z)$ having exactly one occurrence of a special variable [3] $\bullet$ of sort s, where Z is an S-indexed set of special variables s.t. for each $s \in S$, Z_s is infinite. Let $\mathcal{C}_\Gamma[\bullet : s]$ denote the set of all Γ-contexts for sort s, and $var(c)$ the finite set of variables in a context c except $\bullet$. A Γ-context with visible result sort is called a Γ**-experiment**; let $\mathcal{E}_\Gamma[\bullet : s]$ denote the set of all Γ-experiments for sort s, let $\mathcal{C}_{\Gamma,s'}[\bullet : s]$ denote the Γ-contexts of sort s' for sort s, and let $\mathcal{E}_{\Gamma,v}[\bullet : s]$ denote all the Γ-experiments of sort v for sort s. If $c \in \mathcal{C}_{\Gamma,s'}[\bullet : s]$ and $t \in T_{\Sigma,s}(X)$, then $c[t]$ denotes the term in $T_{\Sigma,s'}(var(c) \cup X)$ obtained from c by substituting t for $\bullet$. Further, c generates a map $A_c \colon A_s \to [A^{var(c)} \to A_{s'}]$ on each Σ-algebra A, defined by $A_c(a)(\theta) = a^*_\theta(c)$, where a^*_θ is the unique extension of the map (denoted a_θ) that takes $\bullet$ to a and each $z \in var(c)$ to $\theta(z)$.

The interesting experiments are those of hidden sort, i.e., with $s \in H$.

Example 3.8 Set (Continued). Let Γ contain only the operation $\_ \in \_$. The experiments on sets have the form $N \in \bullet$, where N is any variable of sort *Nat*.

Example 3.9 Stream. (Continued). If Γ contains only the operations *head* and *tail*, then the Γ-experiments on streams have the form $head(tail^n(\bullet))$ for all $n \geq 0$, where $tail^n$ is a short-hand for n applications of *tail*.

We now define a distinctive feature of hidden logics. Two states are equivalent iff they are indistinguishable under Γ-experiments. Notice that it can be quite possible that the generated behavioral equivalence relation is not preserved by some operators in $\Sigma - \Gamma$. In fact, as argued in [13,38] among other places, it is *desired* that some operators are not Γ-behaviorally congruent in order to elegantly deal with nondeterminism in behavioral specifications and to effectively use automated equational reasoning:

Definition 3.10 Given a hidden Σ-algebra A and a hidden subsignature Γ of Σ, the equivalence $a \equiv^\Gamma_\Sigma a'$ iff $A_\gamma(a)(\theta) = A_\gamma(a')(\theta)$ for all Γ-experiments γ and all maps $\theta \colon var(\gamma) \to A$ is called Γ**-behavioral equivalence on** A. We may write $\equiv$ instead of $\equiv^\Gamma_\Sigma$ when Σ and Γ can be inferred from context, and we write $\equiv_\Sigma$ when $\Sigma = \Gamma$. Given any equivalence $\sim$ on A, an operation σ in $\Sigma_{s_1 \ldots s_n, s}$ is **congruent for** $\sim$ iff $A_\sigma(a_1, ..., a_n) \sim A_\sigma(a'_1, ..., a'_n)$ whenever $a_i \sim a'_i$ for $i = 1...n$. An operation σ is Γ**-behaviorally congruent for** A iff it is congruent for $\equiv^\Gamma_\Sigma$. We often write just "congruent" instead of "behaviorally congruent" [4]. A **hidden** Γ**-congruence on** A is an equivalence on A which is the identity on visible sorts and for which each operation in Γ is congruent.

Example 3.11 Set (Continued). Two sets are Γ-behaviorally equivalent iff they have the same elements (they cannot be distinguished by experiments of the form $N \in \bullet$). In the list model of sets, two lists are Γ-behaviorally

[3] These are assumed different from any other variables in a given situation.
[4] A similar notion was given by Padawitz in [34].

equivalent iff they have the same elements, in any order and with any number of multiple occurrences. All the operations are congruent.

Example 3.12 Stream. (Continued). Two streams are Γ-behaviorally equivalent in the standard model of streams if and only if they have the same elements in the same order. Notice that all the operations on lists are also behaviorally congruent.

The following supports several important results in hidden logics, generalizing [23] to operations with more than one hidden argument or that are not behavioral; see [40,38] for a proof. Since final algebras need not exist in this setting [10], existence of a largest hidden Γ-congruence does not depend on them, as it does in coalgebra.

Theorem 3.13 *Given a hidden subsignature* Γ *of* Σ *and a hidden* Σ*-algebra* A*, then* Γ*-behavioral equivalence is the largest hidden* Γ*-congruence on* A*.*

This result, in its special form when $\Gamma = \Sigma$, generalizes the more broadly known (behavioral) equivalence of states in automata and existence of a largest bisimulation in deterministic transition systems (see [23,22] for more details). A first version of such maximality result for behavioral equivalence that we are aware of, but in the restricted case where all the operators in Σ are Γ-behaviorally congruent, can be found in [5].

Definition 3.14 Given hidden Σ-algebra A and Σ-equation $(\forall X)\ t = t'$, say e, A **Γ-behaviorally satisfies** e, written $A \models^{\Gamma}_{\Sigma} e$, iff $\theta(t) \equiv^{\Gamma}_{\Sigma} \theta(t')$ for all $\theta\colon X \to A$. $A \models^{\Gamma}_{\Sigma} E$ iff A Γ-behaviorally satisfies each Σ-equation in E.

When Σ and Γ are clear, we may write $\equiv$ and $\models$ instead of $\equiv^{\Gamma}_{\Sigma}$ and $\models^{\Gamma}_{\Sigma}$, respectively. We only consider unconditional equations in this paper, but most of the theory of hidden algebra also allows conditional equations [23,25,38]. However, some results only allow conditional equations of visible conditions. It would be interesting to know whether the main results presented in this paper could be generalized to arbitrary conditional equations.

Definition 3.15 Given a Σ-equation $(\forall X)\ t = t'$, say e, $\mathcal{E}_\Gamma[e]$ is either the set $\{(\forall X, var(\gamma))\ \gamma[t] = \gamma[t'] \mid \gamma \in \mathcal{E}_\Gamma[\bullet : h]\}$ when the sort h of t, t' is hidden, or the set $\{e\}$ when the sort of t, t' is visible. $\mathcal{E}_\Gamma[E]$ is the set $\bigcup_{e \in E} \mathcal{E}_\Gamma[e]$.

The following result saying that behavioral satisfaction of an equation can be reduced to strict satisfaction of a potentially infinite set of equations is already considered folklore among behaviorists. [38] presents a proof tuned to our setting, where equations with visible conditions are also considered:

Proposition 3.16 $A \models^{\Gamma}_{\Sigma} E$ *if and only if* $A \models_\Sigma \mathcal{E}_\Gamma[E]$.

Definition 3.17 A **behavioral** (or **hidden**) Σ**-specification** (or **-theory**) is a triple (Σ, Γ, E) where Σ is a hidden signature, Γ is a hidden subsignature of Σ, and E is a set of Σ-equations. The operations in $\Gamma - \Sigma\restriction_V$ are called

behavioral. We usually let $\mathcal{B}$, $\mathcal{B}'$, $\mathcal{B}_1$, etc., denote behavioral specifications. A hidden Σ-algebra A **behaviorally satisfies** (or **is a model of**) a behavioral specification $\mathcal{B} = (\Sigma, \Gamma, E)$ iff $A \models^{\Gamma}_{\Sigma} E$, and in this case we write $A \models \mathcal{B}$; we write $\mathcal{B} \models e$ if $A \models \mathcal{B}$ implies $A \models^{\Gamma}_{\Sigma} e$. An operation $\sigma \in \Sigma$ is **behaviorally congruent for** $\mathcal{B}$ iff σ is behaviorally congruent for every $A \models \mathcal{B}$.

All behavioral operations and all hidden constants are behaviorally congruent [40,38], but of course, depending on E, other operations may also be congruent; in fact, all operations are congruent in many practical situations.

Example 3.18 Set (Continued). The following visible equations added to the hidden signature presented before give a behavioral specification of sets:

- $(\forall N : \mathit{Nat})\ N \in \mathit{empty} = \mathit{false}$,
- $(\forall N, M : \mathit{Nat};\ S : \mathit{Set})\ N \in \mathit{add}(M, S) = (N == M)\ \mathit{or}\ (N \in S)$,
- $(\forall N : \mathit{Nat};\ S, S' : \mathit{Set})\ N \in (S \cup S') = (N \in S)\ \mathit{or}\ (N \in S')$, and
- $(\forall N : \mathit{Nat};\ S, S' : \mathit{Set})\ N \in (S \cap S') = (N \in S)\ \mathit{and}\ (N \in S')$.

Example 3.19 Stream. (Continued). The visible and hidden equations below added to the signature of streams, give a behavioral specification:

- $(\forall N : \mathit{Nat};\ S : \mathit{Stream})\ \mathit{head}(N\&S) = N$,
- $(\forall N : \mathit{Nat};\ S : \mathit{Stream})\ \mathit{tail}(N\&S) = S$,
- $(\forall S : \mathit{Stream})\ \mathit{head}(\mathit{odd}(S)) = \mathit{head}(S)$,
- $(\forall S : \mathit{Stream})\ \mathit{tail}(\mathit{odd}(S)) = \mathit{even}(\mathit{tail}(S))$,
- $(\forall S : \mathit{Stream})\ \mathit{head}(\mathit{even}(S)) = \mathit{head}(\mathit{tail}(S))$,
- $(\forall S : \mathit{Stream})\ \mathit{tail}(\mathit{even}(S)) = \mathit{even}(\mathit{tail}(\mathit{tail}(S)))$,
- $(\forall S, S' : \mathit{Stream})\ \mathit{head}(\mathit{zip}(S, S')) = \mathit{head}(S)$, and
- $(\forall S, S' : \mathit{Stream})\ \mathit{tail}(\mathit{zip}(S, S')) = \mathit{zip}(S', \mathit{tail}(S))$.

4 Unhiding

Ordinary algebraic specifications can be associated to behavioral specifications, and special many-sorted algebras can be built from hidden algebras. This section presents all these technical constructions that we generically call "unhiding," and some of their basic properties. We will use mix-fix-like syntactic notation (underscores stay for arguments) to increase the readability of our specifications. We have implemented and experimented with the next concepts and procedures in Maude [12] (see also the next section), but any equational environment could have been used.

4.1 Unhiding a Hidden Signature

A hidden signature can be "unhidden" by associating it the specification of its "experiments" as shown below. It is worth mentioning that unhiding can be

done in different ways and that the major challenge is to get it done right in order to not only prove the theoretical result relating behavioral abstraction with information hiding but also to explain how inductive proofs can be used in practice to show behavioral equivalences. Our first tentative to unhide a behavioral specification was presented in [25] and it essentially tuned a similar construction previously presented in [6] to our hidden logic framework. The construction in [25] was sufficiently good to show the theoretical result, but not to show the practical one. Consequently, we fully agreed with the authors of [6] who stated "however, it should be clear that the encoding of contexts is so complex that this result is of purely theoretical interest." It is the new unhiding procedure presented next that motivated writing the current paper, because it not only allows one to show the information hiding theoretical result, but also gives a mechanical way by which inductive equational proof engines can be used to perform behavioral proofs.

Definition 4.1 If Γ is a hidden signature, let $\tilde{S}$ be the set $S \cup (H \twoheadrightarrow V)$, where S is the set $V \cup H$ and $H \twoheadrightarrow V$ is a set of new sorts of the form $h \twoheadrightarrow v$ where $h \in H$ and $v \in V$. Let $\tilde{\Gamma}$ be the $\tilde{S}$-signature adding to Γ the operations:

- $\sigma\_ : \overline{s} \to h \twoheadrightarrow v$ for all[5] $\sigma : \overline{s}\ h \to v$ in Γ with $h \in H$,
- $\_[\sigma\_] : (h' \twoheadrightarrow v)\ \overline{s} \to h \twoheadrightarrow v$ for all $v \in V$, $\sigma : \overline{s}\ h \to h'$ in[6] Γ s.t. $h, h' \in H$,
- $\_[\_] : (h \twoheadrightarrow v)\ h \to v$ for each $h \in H$ and $v \in V$.

Furthermore, let E_Γ be the set of equations:

- $(\forall Y; x : h)\ \sigma(Y)[x] = \sigma(Y, x)$ for each $\sigma : \overline{s}\ h \to v$, and
- $(\forall Y; Exp : h' \twoheadrightarrow v; x : h)\ Exp[\sigma(Y)][x] = Exp[\sigma(Y, x)]$, for each $\sigma : \overline{s}\ h \to h'$.

The equational specification $(\tilde{\Gamma}, E_\Gamma)$ is called **the unhiding of** Γ.

The sorts $h \twoheadrightarrow v$ stay for "experiments of sort v for sort h". Operations $\sigma\_ : \overline{s} \to h \twoheadrightarrow v$ are curried versions of operations $\sigma : \overline{s}\ h \to v$ in Γ, their role being to produce elementary experiments $\sigma(Y)$, where $Y : \overline{s}$ is an appropriate set of variables; the operations $\_[\sigma\_] : (h' \twoheadrightarrow v)\ \overline{s} \to h \twoheadrightarrow v$ generate experiments for sorts h from experiments for sorts h' by composition with operations $\sigma : \overline{s}\ h \to h'$; operations $\_[\_] : (h \twoheadrightarrow v)\ h \to v$ apply experiments. The first $\tilde{\Gamma}$-equation says that one-operation experiments evaluate as the operation itself, while the second $\tilde{\Gamma}$-equation shows how a composed experiment $Exp[\sigma(Y)]$ works: the state is first plugged into σ and then the whole thing into Exp. Despite its apparently technical formulation, the construction above is very intuitive: it defines experiments and their semantics equationally in a minimal way, avoiding even the occurrence of the artificial variables $\bullet$.

Example 4.2 Set (Continued). The unhiding specification of the specifica-

[5] Since σ can have more than one hidden argument, actually an operation $\sigma^k : \overline{s_k} \to (h_k \twoheadrightarrow v)$ is added for each $\sigma : \overline{s_k}\ h_k \to v$ in Γ and each $k = 1, ..., n$ s.t. $h_k \in H$.
[6] Same observation as in footnote 5.

tion of sets (which contains only the membership attribute) is the following:

```
fmod GAMMA-SET~ is protecting NAT .
  sort Set .
  sort Set->Bool .
  op _in_ : Nat Set -> Bool .
  op _in : Nat -> Set->Bool .
  op _[_] : Set->Bool Set -> Bool .
  var N : Nat .  var S : Set .
  eq N in [S] = N in S .
endfm
```

The sort `Set->Bool` stays for experiments on sets, and `_in : Nat -> Set->Bool` is the curried version of the membership attribute. Since there is no operation of hidden result, there is no operation of the form $\_[\sigma\_]$ added to $\tilde{\Gamma}$. Therefore, there is only one more operation, the "application" $\_[\_]$, and one equation which should be parenthesized like `(N in) [S] = (N in S)`.

Example 4.3 Stream. (Continued). The unhiding specification of the specification $\Gamma = \{head, tail\}$ of streams presented before is the following:

```
fmod GAMMA-STREAM~ is protecting NAT .
  sort Stream .
  sort Stream->Nat .
  op head : Stream -> Nat .
  op tail : Stream -> Stream .
  op head : -> Stream->Nat .
  op _[tail] : Stream->Nat -> Stream->Nat .
  op _[_] : Stream->Nat Stream -> Nat .
  var Exp : Stream->Nat .  var S : Stream .
  eq head[S] = head(S) .
  eq Exp[tail][S] = Exp[tail(S)] .
endfm
```

4.2 Unhiding a Hidden Algebra

Unhiding of a hidden algebra is executed by adding experiments to it. We first need to define experiments locally to a hidden algebra.

Definition 4.4 Given a hidden subsignature Γ of Σ and a hidden Σ-algebra A, a (Γ, A)**-context for sort** s is a term in $T_{\Gamma \cup A}(\{\bullet : s\})$ with exactly one occurrence of $\bullet$. A (Γ, A)**-experiment** is a (Γ, A)-context of visible result. $\mathcal{C}^A_\Gamma[\bullet : s]$ and $\mathcal{E}^A_\Gamma[\bullet : s]$ are the sets of (Γ, A)-contexts and (Γ, A)-experiments.

Notice that the elements in A are added as constants, thus being allowed to be used in contexts and experiments. Obviously, any hidden Σ-algebra A can be regarded as a $(\Gamma \cup A)$-algebra where the operations in Γ are interpreted as in $A\restriction_\Gamma$ and each constant $a \in A$ is interpreted as the element $a \in A$. Conceptually, the contexts in Definition 4.4 are instances of those in Definition

3.7, by replacing their variables different from $\bullet$ with concrete values in A. As expected, the (Γ, A)-experiments generate the behavioral equivalence on A:

Proposition 4.5 *Given a hidden Σ-algebra A and $a, a' \in A_s$, then $a \equiv^{\Gamma}_{\Sigma,s} a'$ if and only if $A_\gamma(a) =_v A_\gamma(a')$ for each $\gamma \in \mathcal{E}^A_{\Gamma,v}[\bullet : s]$ and each $v \in V$.*

One can now unhide any hidden Σ-algebra A into a $\Sigma \cup \tilde{\Gamma}$-algebra by adding (Γ, A)-experiments of sort v for sort h to each carrier $\tilde{A}_{h \twoheadrightarrow v}$. Formally,

Definition 4.6 Given a hidden subsignature Γ of Σ and a hidden Σ-algebra A, let $\tilde{A}$ be the $(\Sigma \cup \tilde{\Gamma})$-algebra[7] defined by:

- $\tilde{A}|_\Sigma = A$, that is $\tilde{A}$ extends A,
- $\tilde{A}_{(h \twoheadrightarrow v)} = \mathcal{E}^A_{\Gamma,v}[\bullet : h]$,
- $\tilde{A}_{\sigma\_} \colon A^{\overline{s}} \to \tilde{A}_{(h \twoheadrightarrow v)}$ is defined by $\tilde{A}_{\sigma\_}(\overline{a}) = \sigma(\overline{a}, \bullet)$, for each $\sigma\_ \colon \overline{s} \to (h \twoheadrightarrow v)$,
- $\tilde{A}_{\_[\sigma\_]} \colon \tilde{A}_{(h' \twoheadrightarrow v)} \times A^{\overline{s}} \to \tilde{A}_{(h \twoheadrightarrow v)}$ is defined by $\tilde{A}_{\_[\sigma\_]}(\gamma, \overline{a}) = \gamma(\sigma(\overline{a}, \bullet))$, for each $\sigma \colon \overline{s}\ h \to h'$, and
- $\tilde{A}_{\_[\_]} \colon \tilde{A}_{(h \twoheadrightarrow v)} \times A_h \to A_v$ is defined by $\tilde{A}_{\_[\_]}(\gamma, a) = A_\gamma(a)$, for each $v \in V$, $h \in H$, $\gamma \in \mathcal{E}^A_{\Gamma,v}[\bullet : h]$, and $a \in A_h$.

The $(\Sigma \cup \tilde{\Gamma})$-algebra $\tilde{A}$ is called **the Γ-unhiding of** A.

The following proposition says that the Γ-unhiding of a hidden Σ-algebra is a model of the unhiding specification of Γ:

Proposition 4.7 *Given $\Gamma \subseteq \Sigma$ and a hidden Σ-algebra A, then $\tilde{A} \models_{\Sigma \cup \tilde{\Gamma}} E_\Gamma$.*

The following proposition is very important because, by Proposition 2.2 via some further results presented in the next sections, it essentially allows one to soundly use inductive proofs on the newly added sorts by unhiding:

Proposition 4.8 *$\tilde{A}$ is $(H \twoheadrightarrow V, Z)$-reachable, for any $(H \cup V)$-indexed set of variables Z.*

4.3 Unhiding a Behavioral Specification

In this subsection we show how a behavioral specification can be automatically unhidden, generating an ordinary specification which is finite whenever the original behavioral specification is finite. Moreover, we show how behavioral proof obligations translate into ordinary equational ones. This is particularly interesting because equational reasoning is not sound in general for behavioral satisfaction because of the behaviorally non-congruent operators.

The following constructions are similar to those in Definition 3.15:

Definition 4.9 If e is a Σ-equation $(\forall X)\ t = t'$ then let $\tilde{e}$ denote either the set of $(\Sigma \cup \tilde{\Gamma})$-equations $\{(\forall X; Exp : h \twoheadrightarrow v)\ Exp[t] = Exp[t'] \mid v \in V\}$ when the

[7] To keep the notation simple, Γ does not occur in the notation of $\tilde{A}$.

sort h of t, t' is hidden, or the set $\{e\}$ when the sort of t, t' is visible. Similarly, let $\tilde{E}$ be the set $\bigcup_{e \in E} \tilde{e}$; then $\tilde{\mathcal{B}} = (\Sigma \cup \tilde{\Gamma}, \tilde{E} \cup E_\Gamma)$ is **the unhiding of $\mathcal{B}$**.
Notice that $\tilde{\mathcal{B}}$ is finite whenever $\mathcal{B}$ is finite.

Example 4.10 Set (Continued). The unhiding of the behavioral sets is:

```
fmod SET~ is extending GAMMA-SET~ .
  op empty : -> Set .
  op add : Nat Set -> Set .
  ops (_U_) (_&_) : Set Set -> Set .
  vars N M : Nat .  vars S S' S'' : Set .
  eq N in empty = false .
  eq N in (S U S') = (N in S) or (N in S') .
  eq N in add(M, S) = (N == M) or (N in S) .
  eq N in (S & S') = (N in S) and (N in S') .
endfm
```

Notice that the unhiding of Γ, `GAMMA-SET~`, was imported. Since all the equations are of visible sort, they are left unchanged.

Example 4.11 Stream (Continued). The unhiding of behavioral streams is:

```
fmod STREAM~ is extending GAMMA-STREAM~ .
  op  _&_  : Nat Stream -> Stream .
  ops odd even : Stream -> Stream .
  op  zip : Stream Stream -> Stream .
  var N : Nat .  vars S S' : Stream .  var Exp : Stream->Nat .
  eq head(N & S) = N .
  eq Exp[tail(N & S)] = Exp[S] .
  eq head(odd(S)) = head(S) .
  eq Exp[tail(odd(S))] = Exp[even(tail(S))] .
  eq head(even(S)) = head(tail(S)) .
  eq Exp[tail(even(S))] = Exp[even(tail(tail(S)))] .
  eq head(zip(S,S')) = head(S) .
  eq Exp[tail(zip(S,S'))] = Exp[zip(S',tail(S))] .
endfm
```

The last equation, for example, intuitively says that for any experiment `Exp` and any streams `S` and `S'`, the experiment `Exp` returns the same element when evaluated on the streams `tail(zip(S,S'))` and `zip(S',tail(S))`.

Proposition 4.12 *Given a behavioral specification $\mathcal{B} = (\Sigma, \Gamma, E)$, a Σ-equation e, and a hidden Σ-algebra A, then*
(i) $A \models^{\Gamma}_{\Sigma} e$ *iff* $\tilde{A} \models_{\Sigma \cup \tilde{\Gamma}} \tilde{e}$,
(ii) $A \models \mathcal{B}$ *iff* $\tilde{A} \models \tilde{\mathcal{B}}$, *and*
(iii) $\tilde{\mathcal{B}} \models \tilde{e}$ *implies* $\mathcal{B} \models e$.

This proposition suggests that in order to show that e is a behavioral consequence of $\mathcal{B}$, it suffices to show that $\tilde{e}$ is an equational consequence of $\tilde{\mathcal{B}}$.

As shown next, this simple proof technique is too weak in practical situations. Note that (iii) cannot be an equivalence because it would otherwise provide a complete calculus for behavioral satisfaction, which is incomplete [10].

5 Practical Importance: Context Induction

If one wants to prove $\texttt{SET} \models (\forall \texttt{S,S':Set})\ \texttt{S U S'} = \texttt{S' U S}$ by (iii) in Proposition 4.12, then one is stuck since one has to prove by ordinary equational reasoning $\texttt{SET}^{\sim} \models (\forall \texttt{S,S':Set; Exp:(Set->Bool)})\ \texttt{Exp[S U S']} = \texttt{Exp[S' U S]}$, which is impossible. Some kind of induction on contexts is needed.

Definition 5.1 Given behavioral specification $\mathcal{B} = (\Sigma, \Gamma, E)$ and Σ-equation e, then $\tilde{\mathcal{B}}$ $(H \twoheadrightarrow V, Z)$**-inductively satisfies** e, written $\tilde{\mathcal{B}} \models_{Ind(H \twoheadrightarrow V,Z)} \tilde{e}$, if and only if $T_{\Sigma \cup \tilde{\Gamma}}(Z)/_{\tilde{E} \cup E_\Gamma} \models \tilde{e}$.

The definition above weakens satisfaction to only a special model of $\tilde{\mathcal{B}}$. However, this model has good properties. First, since it is a free model and there are no variables of sorts $(h \twoheadrightarrow v)$ in Z, proofs by induction on sorts in $H \twoheadrightarrow V$ are valid in $T_{\Sigma \cup \tilde{\Gamma}}(Z)/_{\tilde{E} \cup E_\Gamma}$; in particular, one can prove statements like $T_{\Sigma \cup \tilde{\Gamma}}(Z)/_{\tilde{E} \cup E_\Gamma} \models_{\Sigma \cup \tilde{\Gamma}} (\forall z : (h \twoheadrightarrow v), X)\ z[t] = z[t']$ by structural induction on $z : (h \twoheadrightarrow v)$. Second, for any other model A' of $\tilde{\mathcal{B}}$, it is the case that any map $\tau : Z \to A'$ uniquely extends to a morphism $\tau : T_{\Sigma \cup \tilde{\Gamma}}(Z)/_{\tilde{E} \cup E_\Gamma} \to A'$.

Proposition 5.2 $\tilde{\mathcal{B}} \models_{Ind(H \twoheadrightarrow V,Z)} \tilde{e}$ *implies* $\mathcal{B} \models e$.

Proposition 5.2 suggests the following procedure to do behavioral proofs: 1) generate $\tilde{B}$ and $\tilde{e}$; 2) show $\tilde{\mathcal{B}} \models_{Ind(H \twoheadrightarrow V,Z)} \tilde{e}$ either manually or using an inductive theorem prover; 3) conclude $\mathcal{B} \models e$. We next analyze examples.

Example 5.3 Set (Continued). According to proposition above, distributivity of intersection and union reduces to showing that $\texttt{SET}^{\sim}$ inductively satisfies $(\forall \texttt{S,S',S'':Set;Exp:Set->Bool})\ \texttt{Exp[S\&(S' U S'')]} = \texttt{Exp[(S \& S')U(S \& S'')]}$, which can be shown with the Maude proof score

```
fmod DISTRIBUTIVITY-PROOF is protecting SET~ .
  ops s s' s'' : -> Set .
  op exp : -> Set->Bool .
  op n : Nat .
  eq exp = n in .
endfm
red exp[s & (s' U s'')] == exp[(s & s') U (s & s'')] .
***> should be true
```

using the theorem of constants and (degenerated) induction on experiments.

Example 5.4 Stream (Continued). The behavioral proofs for sets are simple because of the oversimplified structure of experiments. However, proofs by context induction become much harder, often impractical, when experiments are complex. The next proof shows how nontrivial the task can be even for

relatively simple contexts, such as those of streams. The reader is encouraged to compare this with the elegant and completely automatic proofs by circular coinductive rewriting of the same property and many others in [20,38].

We next prove that $zip(odd(S), even(S))$ is behaviorally equivalent to S, for any stream S. As before, it suffices to show that `STREAM~` inductively satisfies (∀`S:Stream; Exp:Stream->Nat`) `Exp[zip(odd(S), even(S))]` = `Exp[S]`.

We need some auxiliary lemmas. First, let us show the congruence of `zip`. Let $\mathcal{P}$ be the predicate on experiments such that $\mathcal{P}$(`Exp`) if and only if `STREAM~` satisfies (∀`Exp:Stream->Nat`) `Exp[zip(s1,s2)]` = `Exp[zip(s1',s2')]` for any behaviorally equivalent streams `s1` and `s1'`, and any behaviorally equivalent streams `s2` and `s2'`. We show that $\mathcal{P}$(`Exp`) holds for all experiments `Exp` by structural induction. $\mathcal{P}$(`head`) holds because `head(s1)` is equal to `head(s1')` for any behaviorally equivalent `s1` and `s2`. Assume $\mathcal{P}$(`exp`) for some experiment `exp`, and let us fix some `s1`, `s1'`, `s2` and `s2'` as above; then `exp[zip(s2, tail(s1))]` equals `exp[zip(s2', tail(s1'))]` because `tail` is congruent, and further one can easily show now by rewriting that `exp[tail][zip(s1, s2)]` equals `exp[tail][zip(s1', s2')]`; so $\mathcal{P}$(`exp[tail]`) also holds. The following is the Maude proof score:

```
fmod ZIP-CONG-PROOF is protecting STREAM~ .
  ops s1 s1' s2 s2' : -> Stream .
  ops exp : -> Stream->Nat .
  eq head(s1) = head(s1') .
  eq exp[zip(s2, tail(s1))] = exp[zip(s2', tail(s1'))] .
endfm
red      head[zip(s1, s2)] ==      head[zip(s1', s2')] .
red exp[tail][zip(s1, s2)] == exp[tail][zip(s1', s2')] .
***> should both be true
```

Therefore, `zip` preserves the behavioral equivalence, in particular the equations of the initial behavioral specification of streams. We only need three instances:

```
fmod LEMMAS is protecting STREAM~ .
 vars S S' : Stream .  var Exp : Stream->Nat .
 eq Exp[zip(S',tail(odd(S)))]  = Exp[zip(S',even(tail(S)))] .
 eq Exp[zip(tail(odd(S)),S')]  = Exp[zip(even(tail(S)),S')] .
 eq Exp[zip(S',tail(even(S)))] = Exp[zip(S',even(tail(tail(S))))] .
```

Notice that `STREAM~` is not a Church-Rosser rewriting system because the term `head[tail(odd(S))]` admits the normal forms `head(tail(odd(S)))` and `head(tail(tail(S)))`. Therefore, if one uses a rewriting based equational prover like Maude, then one may need to add some auxiliary lemmas [8]. We need only one in our proof:

```
 eq Exp[tail(tail(zip(S,S')))] = Exp[tail(zip(S',tail(S)))] .
endfm
```

[8] Or alternatively, run a Church-Rosser completion procedure, such as Knuth-Bendix.

There is one more lemma needed, relating `zip` and `even`, which we first prove:

```
fmod ZIP-EVEN-LEMMA-PROOF is protecting LEMMAS .
  op s : -> Stream .
  op exp : -> Stream->Nat .
  var S : Stream .
  eq exp[zip(even(S), even(tail(S)))] = exp[tail(S)] .
endfm
red      head[zip(even(s), even(tail(s)))] ==      head[tail(s)] .
red exp[tail][zip(even(s), even(tail(s)))] == exp[tail][tail(s)] .
***> should both be true
```

and then append to the other lemmas:

```
fmod ZIP-EVEN-LEMMA is protecting LEMMAS .
  var S : Stream .  var Exp : Stream->Nat .
  eq Exp[zip(even(S), even(tail(S)))] = Exp[tail(S)] .
endfm
```

We can now inductively prove the initial result:

```
fmod ZIP-LEMMA-PROOF is protecting ZIP-EVEN-LEMMA .
  op s : -> Stream .
  op exp : -> Stream->Nat .
endfm
red      head[zip(odd(s), even(s))] ==      head[s] .
red exp[tail][zip(odd(s), even(s))] == exp[tail][s] .
***> should both be true
```

The inductive technique used in the examples above was nothing but what is called *context induction* [27] (see also [4] for related work). In fact, any proof technique for the ordinary algebraic specification $\tilde{\mathcal{B}}$ is allowed, as far as it is sound at least for the models $\tilde{A}$ associated to hidden algebras. As the reader probably guesses, the inductive proof in the example above needed significant human intervention. Even if the whole inductive proof can be automated in some complicated way, we encourage the readers interested in automation of behavioral reasoning to also check out *circular coinductive rewriting*, which is implemented in BOBJ [38]. We have not encountered any behavioral property that can be proved by context induction but not by circular coinductive rewriting automatically yet.

6 Behavioral Abstraction is Information Hiding

We now introduce the main theoretical result of the paper, namely that, semantically, behavioral abstraction is a special case of information hiding:

Theorem 6.1 *Given a behavioral specification $\mathcal{B} = (\Sigma, \Gamma, E)$ and a hidden Σ-algebra A, if $\Sigma\Box\tilde{\mathcal{B}}$ is the ordinary equational Σ-theory consisting of all the Σ-theorems of $\tilde{\mathcal{B}}$ then*

(i) $A \models\!\!\models \mathcal{B}$ *iff* $A \models \Sigma\Box\tilde{\mathcal{B}}$, *and*

(ii) *In the loose-data hidden algebra case,* $\mathcal{B}$ *and* $\Sigma\Box\tilde{\mathcal{B}}$ *have the same models.*

The proof of Theorem 6.1 follows by induction on experiments, using the fact that the unhiding models $\tilde{A}$ are $(H \twoheadrightarrow V, Z)$-reachable (see Proposition 4.8).

7 Conclusion

By adding machinery for experiments, use it and then hide it, we showed how any behavioral Σ-specification $\mathcal{B}$ can be "unhidden" to an ordinary algebraic specification $\tilde{\mathcal{B}}$ over a larger signature, such that a model behaviorally satisfies $\mathcal{B}$ if and only if it satisfies, in the ordinary sense, the Σ-theorems of $\tilde{\mathcal{B}}$. The construction of $\tilde{\mathcal{B}}$ is algorithmic and finite when $\mathcal{B}$ is finite. The practical aspect of our procedure is that we have developed a technique by which one can safely use induction and equational deduction in $\tilde{\mathcal{B}}$ to reason about behavioral equality in $\mathcal{B}$, despite the fact that neither of those is sound in $\mathcal{B}$. An interesting direction of future work is to use automated inductive theorem provers to show behavioral equivalences and to compare their results to BOBJ's circular coinductive rewriting. On the theoretical side, the relationship between the two extensions of algebraic specifications can lead to Craig interpolation results for hidden logics.

References

[1] J. Bergstra, J. Heering, and P. Klint. Module algebra. *Journal of the ACM*, 37(2):335–372, 1990.

[2] J. Bergstra and J.V. Tucker. Equational specifications, complete rewriting systems, and computable and semicomputable algebras. *Journal of the ACM*, 42(6):1194–1230, 1995.

[3] G. Bernot, M. Bidoit, and T. Knapik. Observational specifications and the indistinguishability assumption. *TCS*, 139(1-2):275–314, 1995.

[4] N. Berregeb, A. Bouhoula, and M. Rusinowitch. Observational proofs with critical contexts. In *Proceedings of FASE'98*, volume 1382 of *LNCS*. 1998.

[5] M. Bidoit and R. Hennicker. Proving behavioural theorems with standart first-order logic. In *Algebraic and Logic Programming (ALP'94)*, volume 850 of *LNCS*, pages 41–58, 1994.

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

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

[8] M. Bidoit and R. Hennicker. Observer complete definitions are behaviourally coherent. In *OBJ/CafeOBJ/Maude at FM'99*, pages 83–94. Theta, 1999.

[9] R. Burstall and R. Diaconescu. Hiding and behaviour: an institutional approach. In *A Classical Mind: Essays in Honour of C.A.R. Hoare*, pages 75–92. Prentice Hall, 1994.

[10] S. Buss and G. Roşu. Incompleteness of behavioral logics. In *Proceeding of CMCS'00*, volume 33 of *ENTCS*, pages 61–79. Elsevier, 2000.

[11] C. Cîrstea. Semantic constructions for hidden algebra. In *Recent Trends in Algebraic Development Techniques*, volume 1589 of *LNCS*. Springer, 1999.

[12] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In *Proceedings of WRLA*, volume 4 of *ENTCS*. Elsevier, 1996.

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

[14] R. Diaconescu and K. Futatsugi. Behavioral coherence in object-oriented algebraic specification. *J. of Universal Computer Science*, 6(1):74–96, 2000.

[15] R. Diaconescu, J. Goguen, and P. Stefaneas. Logical support for modularization. In *Logical Environments*, pages 83–130. Cambridge, 1993.

[16] V. Giarrantana, F. Gimona, and U. Montanari. Observability concepts in abstract data specifications. In *Proceedings of MFCS*, vol. 45 of *LNCS*. 1976.

[17] J. Goguen. Types as theories. In *Topology and Category Theory in Computer Science*, pages 357–390. Oxford, 1991.

[18] J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, January 1992.

[19] J. Goguen and R. Diaconescu. Towards an algebraic semantics for the object paradigm. In *Proceedings of WADT*, volume 785 of *LNCS*. Springer, 1994.

[20] J. Goguen, K. Lin, and G. Roşu. Circular coinductive rewriting. In *Proceedings of Automated Software Engineering 2000*, pages 123–131. IEEE, 2000.

[21] J. Goguen, K. Lin, and G. Roşu. Behavioral and coinductive rewriting. In *Proceedings of WRLA'01*, volume 36 of *ENTCS*, pages 1–22. Elsevier, 2001.

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

[23] J. Goguen and G. Malcolm. A hidden agenda. *TCS*, 245(1):55–101, 2000.

[24] J. Goguen and J. Meseguer. Universal realization, persistent interconnection and implementation of abstract modules. In *Proceedings of ICALP*, volume 140 of *LNCS*, pages 265–281. Springer, 1982.

[25] J. Goguen and G. Roşu. Hiding more of hidden algebra. In *Proceeding of FM'99*, volume 1709 of *LNCS*, pages 1704–1719. Springer, 1999.

[26] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In *Software Engineering with OBJ: algebraic specification in action*, pages 3–167. Kluwer, 2000.

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

[28] R. Hennicker and M. Bidoit. Observational logic. In *Proceedings of AMAST'98*, volume 1548 of *LNCS*, pages 263–277. Springer, 1999.

[29] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the European Association for Theoretical Comp. Science*, 62:222–259, 1997.

[30] D. Lucanu, O. Gheorghieş, and A. Apetrei. Bisimulation and hidden algebra. In *Proceedings of CMCS'99*, volume 19 of *ENTCS*, pages 213–232. 1999.

[31] M. Majster. Limits of the algebraic specification of abstract data types. *SIGPLAN Notices*, pages 37–42, October 1977.

[32] J. Meseguer and J. Goguen. Initiality, induction and computability. In *Algebraic Methods in Semantics*, pages 459–541. Cambridge, 1985.

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

[34] P. Padawitz. Towards the one-tiered design of data types and transition systems. In *Proceedings of WADT'97*, volume 1376 of *LNCS*, pages 365–380. Springer, 1998.

[35] D. Parnas. Information distribution aspects of design methodology. *Information Processing*, 71:339–344, 1972.

[36] H. Reichel. Behavioural equivalence – a unifying concept for initial and final specifications. In *Proceedings of the 3rd Hungarian Computer Science Conference*. Akademiai Kiado, 1981.

[37] H. Reichel. Behavioural validity of conditional equations in abstract data types. In *Contributions to General Algebra 3*. Teubner, 1985.

[38] G. Roşu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000.

[39] G. Roşu. Equational axiomatizability for coalgebra. *Theoretical Computer Science*, 260(1-2):229–247, 2001.

[40] G. Roşu and J. Goguen. Hidden congruent deduction. In *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *LNAI*. Springer, 2000.

[41] D. Sannella and A. Tarlecki. On observational equivalence and algebraic specification. *Journal of Computer and System Science*, 34:150–178, 1987.