Journal of TCS, Volume 327(1-2), pp 197-221. 2004

Behavioral Abstraction is Hiding Information

Grigore Roşu ⋆

Department of Computer Science
University of Illinois at Urbana-Champaign, USA

Abstract

We show that for any behavioral Σ-specification $\mathbb{B}$ there is an ordinary algebraic specification $\tilde{\mathbb{B}}$ over a larger signature, such that a model behaviorally satisfies $\mathbb{B}$ iff it satisfies, in the ordinary sense, the Σ-theorems of $\tilde{\mathbb{B}}$. 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 $\mathbb{B}$ and produces a finite $\tilde{\mathbb{B}}$. 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.

Key words: Algebraic specification, behavioral equivalence, information hiding.

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 [45] 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 [38] suggested that

⋆ This work is partly supported by the joint NSF/NASA grant CCR-0234524.

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 [39]) 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. 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 [19] present logic paradigm independent (or *institutional* [22]) 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 and many others [21,23,27,26,51,29], *observational logic* in works by Hennicker, Bidoit and many others [34,8,4,3], *swinging types*[1] in works by Padawitz [42–44], *coherent hidden algebra* in Diaconescu [18], *hidden logic* in Roşu [48], 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[2], 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 to Montanari 1976 [20], Reichel 1981 [46,47], Goguen and Meseguer 1982 [28], and to Sannella and Tarlecki 1987 [54].

A closely related and elegant subject is *coalgebra* (for example see Jacobs and Rutten [35]): 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 [24–26,51]. From now on in this paper we shall use the terminology of hidden logic as in [48]. Even though hidden logic specifications can go beyond the capabilities of standard coalgebra by allowing operations with multiple hidden arguments[3] (which lead to the nonexistence of final models) and operators that do not preserve the behavioral equivalence, it is fair to say that a major limitation of the current hidden algebra variants is

[1] "Swinging types" is a generic notion including behavioral abstraction.

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

[3] Extensions of coalgebra with binary methods are also investigated, e.g., by Tews [57,56,55].

their lack of providing explicit support for "sum codomain sorts". The models of hidden logic specifications are special standard many-sorted algebras, so they are by their nature deterministic.

A *behavioral* Σ*-specification*, usually written $\mathbb{B}, \mathbb{B}', \mathbb{B}_1, ...$, 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 power of information hiding. More precisely, we show that for any behavioral Σ-specification $\mathbb{B} = (\Sigma, \Gamma, E)$ there is some specification $\tilde{\mathbb{B}} = (\Sigma', E')$ for some $\Sigma \subseteq \Sigma'$, called the *unhiding* of $\mathbb{B}$, such that a hidden Σ-algebra behaviorally satisfies $\mathbb{B}$ iff it strictly satisfies $\Sigma \Box \tilde{\mathbb{B}}$, which is the Σ-theory of all Σ-theorems of $\tilde{\mathbb{B}}$. Moreover, E' is finite whenever E is finite, and $\tilde{\mathbb{B}}$ can be generated automatically from $\mathbb{B}$. Even further, $\tilde{\mathbb{B}}$ is generated in such a way that inductive equational theorem provers can be used to prove behavioral equivalence in $\mathbb{B}$.

The general idea of unhiding in this paper is taken from [29], 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 [33]. Previous work by Bidoit, Hennicker [7] and Mikami [40] 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 $\varphi : \Sigma \to \Sigma'$ 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 the "satisfaction condition property" [22] that in the context above, $A' \models_{\Sigma'} \varphi(e)$ iff $A'{\restriction_\varphi} \models_\Sigma e$.

We use Maude [15,14] equational notation in the two examples that we follow in the paper. We find it very intuitive so we do not describe it here, mentioning that it is almost identical to the OBJ notation [31].

2 Reachability and Induction

Induction is *not* sound for all the models of a specification, but only for the *reachable* ones, namely those for which the unique morphism from the initial model is surjective. In other words, in a reachable model each element can be (not necessarily uniquely) labeled by a ground term. However, in this paper we are interested in induction on a *subset of sorts*, namely those that will correspond to *experiments* and that will be generated later by *unhiding*. Therefore, we need a more general approach to reachability and induction.

Definition 1 *Let (S, Σ) be a many-sorted signature. Given $Q \subseteq S$ and an $(S - Q)$-indexed set Z of variables*[4]*, a Σ-algebra A is (Q, Z)-**reachable** if and only if for all $q \in Q$ and $a \in A_q$ there is some map $\theta : Z \to A$ and some term $\gamma \in T_{\Sigma,q}(Z)$ of sort q over variables in Z such that $\theta(\gamma) = a$.*

Intuitively, A is (Q, Z)-reachable if and only if any element in A of sort $q \in Q$ can be generated within A by starting with some elements of sorts in $S-Q$ and by applying the interpretations in A of the operations in Σ. The usual notion of reachability is a special case of the above when $Q = S$. The importance of (Q, Z)-reachability is captured by the following:

Proposition 2 *In the context of Definition 1, if A is (Q, Z)-reachable and if $\mathbb{P}^T$ and $\mathbb{P}^A$ are Q-indexed predicates on $T_\Sigma(Z)\restriction_Q$ and $A\restriction_Q$, respectively, such that for any $q \in Q$ and $\gamma \in T_{\Sigma,q}(Z)$ it is the case that $\mathbb{P}^T(\gamma)$ implies $\mathbb{P}^A(\theta(\gamma))$ for any $\theta : Z \to A$, then $\mathbb{P}^T = T_\Sigma(Z)\restriction_Q$ implies $\mathbb{P}^A = A\restriction_Q$.*

Proof: Let $q \in Q$ and let $a \in A_q$. By Definition 1, there is some $\gamma \in T_{\Sigma,q}(Z)$ such that $\theta(\gamma) = a$. Since $\mathbb{P}^T(\gamma)$ holds, it follows that $\mathbb{P}^A(\theta(\gamma))$ also holds, that is, that $\mathbb{P}^A(a)$ holds. Therefore, $\mathbb{P}^A = A\restriction_Q$. □

In practice, $\mathbb{P}^A$ is a property that one wants to show for all the elements of sorts in Q of a model A. The proposition above says that if A is (Q, 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 Q. This proof can be done by induction or by any other proof technique on term algebras. In the context of this paper, Q will refer to experiments and properties will be of the form "two elements are indistinguishable under the given experiment". We shall later see how proofs by context induction can be done in this style.

[4] By abuse of language, Z also denotes the S-indexed set with $Z_q = \emptyset$ for all $q \in Q$.

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 [21] and developed further in [23,9,27,49,51,18,13,26,29,37,24,25,48] among other places. Two systems, CafeOBJ [17] and BOBJ [24,25,48], supporting behavioral specification and reasoning have been implemented, both extending OBJ [31]. A comprehensive presentation of hidden algebra can be found in [48]. 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* [48] refer to close relatives of hidden algebra, including both fixed-data and loose-data variants.

3.1 Hidden Signature

Definition 3 *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 $\Gamma{\restriction_V} = \Sigma{\restriction_V}$.*

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).

Definition 4 *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**.

Example 5 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 6 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 $zip : Stream \times Stream \to Stream$. ■

3.2 Hidden Algebra

Definition 7 *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 $A\restriction_{(\Sigma\restriction_V)} = D$.*

The first definition of hidden algebra was fixed-data [21], 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 [11,49]; 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 includes both loose-data and fixed-data approaches.

Example 8 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 known. 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; multiple occurrences of elements are allowed. ■

Example 9 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.

3.3 Contexts and Experiments

An *experiment* is an observation of an attribute of a system after it has been perturbed; 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 design 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 also for other important reasons that will be discussed in Subsection 3.7.

Definition 10 *A* **Γ-context for sort** *s is a term in $T_\Gamma(\{\bullet : s\} \cup Z)$ having exactly one occurrence of a special variable*[5] *$\bullet$ of sort s, where Z is an S-indexed set of special variables such that Z_s is infinite for each $s \in S$. Let $\mathbb{C}_\Gamma[\bullet : s]$ 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 $\mathbb{E}_\Gamma[\bullet : s]$ denote the set of all Γ-experiments for sort s, let $\mathbb{C}_{\Gamma,s'}[\bullet : s]$ denote the Γ-contexts of sort s' for sort s, and let $\mathbb{E}_{\Gamma,v}[\bullet : s]$ denote all the Γ-experiments of sort v for sort s.*

The interesting experiments are those for hidden sorts $s \in H$. Experiments for visible sorts are allowed just to smooth the presentation. We sometimes say that an experiment or a context for sort s is *appropriate* for terms or equations of sort s. Contexts can be "applied" as follows:

Definition 11 *If $c \in \mathbb{C}_{\Gamma,s'}[\bullet : s]$ 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 $\theta(z)$.*

Example 12 Set (Continued). We let Γ contain only the membership operation $\_ \in \_$. The experiments on sets then have the form $N \in \bullet$, where N is any variable of sort *Nat*. ■

Example 13 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*. ■

[5] These are assumed different from any other variables in a given situation.

3.4 Behavioral Equivalence

We now define the important notion of behavioral equivalence. Two states are equivalent if and only if 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 $\Sigma - \Gamma$; this aspect will be discussed in more depth in Subsection 3.7.

Definition 14 *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"*[6]*. 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 15 Set (Continued). Two sets are Γ-behaviorally equivalent if and only if they have the same elements, that is, they cannot be distinguished by experiments of the form $N \in \bullet$. In the list model of sets, two lists are Γ-behaviorally equivalent iff they have the same elements, in any order and with any number of multiple occurrences. All the operations are congruent. ■

Example 16 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 streams are also behaviorally congruent. ■

The following supports several important results in hidden logics, generalizing [27] to operations with more than one hidden argument or that are not behavioral; the interested reader is referred to [51,48] for a proof. Since final algebras may not exist in this setting [10], the existence of a largest hidden Γ-congruence does not depend on them.

Theorem 17 *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 [27,26] for more details). A first version of such a maximality result for behavioral equivalence that

[6] A similar notion was given by Padawitz in [43].

we are aware of, but in the restricted case where all the operators in Σ are Γ-behaviorally congruent, can be found in [5].

3.5 Behavioral Satisfaction

Definition 18 *Given a hidden Σ-algebra A and a Σ-equation $(\forall X)\ t = t'$, say e, then A* **Γ-behaviorally satisfies** *e, written $A \models^{\Gamma}_{\Sigma} e$, if and only if $\theta(t) \equiv^{\Gamma}_{\Sigma} \theta(t')$ for all $\theta\colon X \to A$. $A \models^{\Gamma}_{\Sigma} E$ if and only if 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 [27,29,48]. However, some results only allow conditional equations of visible conditions. It would be interesting to know to what extent the main results presented in this paper could be generalized to arbitrary conditional equations.

Definition 19 *Given a Σ-equation $(\forall X)\ t = t'$, say e, $\mathbb{E}_{\Gamma}[e]$ is either the set $\{(\forall X, var(\gamma))\ \gamma[t] = \gamma[t'] \mid \gamma \in \mathbb{E}_{\Gamma}[\bullet : h]\}$ when the sort h of t, t' is hidden, or the set $\{e\}$ when the sort of t, t' is visible. $\mathbb{E}_{\Gamma}[E]$ is the set $\bigcup_{e \in E} \mathbb{E}_{\Gamma}[e]$.*

The following result says that behavioral satisfaction of an equation can be reduced to strict satisfaction of a potentially infinite set of equations. [48] presents a proof of a more general result, where conditional equations with visible conditions are also considered:

Proposition 20 *$A \models^{\Gamma}_{\Sigma} E$ if and only if $A \models_{\Sigma} \mathbb{E}_{\Gamma}[E]$.*

Proof: It suffices to show that $A \models^{\Gamma}_{\Sigma} e$ if and only if $A \models_{\Sigma} \mathbb{E}_{\Gamma}[e]$, for any equation e, say $(\forall X)\ t = t'$. First, suppose that $A \models^{\Gamma}_{\Sigma} e$ and let us consider any equation $(\forall X, var(\gamma))\ \gamma[t] = \gamma[t']$ in $\mathbb{E}_{\Gamma}[e]$. Let $\theta'\colon X \cup var(\gamma) \to A$ be any map. Since t, t' contain only variables in X, it follows that $\theta' \restriction_X (t) \equiv \theta' \restriction_X (t')$. Further, by the definition of behavioral equivalence, $A_\gamma(\theta' \restriction_X (t))(\theta' \restriction_{var(\gamma)}) = A_\gamma(\theta' \restriction_X (t'))(\theta' \restriction_{var(\gamma)})$. But notice that $A_\gamma(\theta' \restriction_X (t))(\theta' \restriction_{var(\gamma)}) = \theta'(\gamma[t])$ and similarly for t', so $\theta'(\gamma[t]) = \theta'(\gamma[t'])$. Conversely, suppose that $A \models_{\Sigma} \mathbb{E}_{\Gamma}[e]$ and let $\theta\colon X \to A$ be a map. Let γ be any Γ-experiment appropriate for t, t' and let $\theta_\gamma\colon var(\gamma) \to A$ be any map. Let $\theta'\colon X \cup var(\gamma) \to A$ be the map such that $\theta' \restriction_X = \theta$ and $\theta' \restriction_{var(\gamma)} = \theta_\gamma$. Then $\theta'(\gamma[t]) = \theta'(\gamma[t'])$, and since $\theta'(\gamma[t]) = A_\gamma(\theta(t))(\theta_\gamma)$ and similarly for t', one gets that $\theta(t) \equiv \theta(t')$. □

3.6 Behavioral Specification

Definition 21 *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* $\mathbb{B}$*,* $\mathbb{B}'$*,* $\mathbb{B}_1$*, etc., denote behavioral specifications. A hidden* Σ*-algebra* A **behaviorally satisfies** *(or* **is a model of***) a behavioral specification* $\mathbb{B} = (\Sigma, \Gamma, E)$ *iff* $A \models^{\Gamma}_{\Sigma} E$*, and in this case we write* $A \models \mathbb{B}$*; we write* $\mathbb{B} \models e$ *if* $A \models \mathbb{B}$ *implies* $A \models^{\Gamma}_{\Sigma} e$*. An operation* $\sigma \in \Sigma$ *is* **behaviorally congruent for** $\mathbb{B}$ *iff* σ *is behaviorally congruent for every* $A \models \mathbb{B}$*.*

All behavioral operations and all hidden constants are behaviorally congruent [51,48], but of course, depending on E, other operations may also be congruent; in fact, all operations are congruent in many practical situations.

Example 22 Set (Continued). The following visible equations added to the hidden signature presented before give a behavioral specification of sets:

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

Example 23 Stream (Continued). The visible and hidden equations below added to the signature of streams, give a behavioral specification:

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

3.7 On Behaviorally Non-Congruent Operations

Even though the examples in this paper contain only behaviorally congruent operations, one of our major goals during the development of the translation results presented in the sequel was to also support operations which do not preserve the behavioral equivalence. Of course, our results are totally applicable in settings where all operations are behaviorally congruent. In this subsection we briefly discuss the reasons for which we believe that behaviorally non-

congruent operations are useful in practice and for which they are supported both by BOBJ and CafeOBJ. Note first that behaviorally non-congruent operators are theoretically and technically inconvenient, because they lead to a series of complications, such as:

- Equational deduction becomes *unsound* for behavioral satisfaction, because the congruence inference rule is not sound for behaviorally non-congruent operations. To solve this problem, a modified, behaviorally sound equational inference system is given in [51].
- As a consequence of the above, term rewriting as a procedure for automatic behavioral proving is not sound either. Special *behavioral rewriting* techniques had to be devised [17,50] and implemented in BOBJ and CafeOBJ.
- One *cannot* build the quotient of a hidden algebra by the behavioral equivalence because the latter is not a congruence, so many algebraic theoretical properties cannot be proved elegantly. Fortunately, the coinductive proof principle is not affected (thanks to Theorem 17).

There are two major practical reasons for which behaviorally non-congruent operations are accepted and tool supported in the hidden algebra world:

Behavioral underspecification. It is often the case during the development process of a system, that one wants to or can specify only a relevant part of a system, with the intention that more properties will be added later in the development process. This way, one does not commit on identifying one unique model at an early stage of the system design, but rather starts with a large class of models and refines it incrementally as the system design evolves. Underspecification resembles nondeterminism in that it leaves open the possibility of choosing among several admissible models, but it differs from nondeterminism in that the latter admits choices within a model. For more information on different approaches to nondeterminism and underspecification, the interested reader is referred to [58]. In the context of behavioral abstraction, underspecification occurs not only at the level of choice among models, but also at the level of the behavioral equivalence within a given model.

Let us describe this phenomenon on an example inspired from [27,29]. Suppose that one wants to specify a random number generator for a distributed system in a producer-consumer style, as a process that generates numbers and stores them (e.g., in a stack or a buffer), where numbers are then consumed by other processes. Its signature can contain a visible sort *Nat*, a hidden sort *State*, a hidden constant *empty* of sort *State* which is the initial state of the generator, an attribute $get : State \to Nat$ returning the currently available random number, a method $remove : State \to State$ discarding that number, and a method $generate : State \to State$ producing a new number. Client processes use the attribute *get* to obtain new numbers, and then the producer process automatically removes those numbers using the *remove* method in

order to avoid reporting the same number to different clients. Since one has a wide range of satisfactory possibilities to implement such a number generator, for example using the time of the microprocessor or other external stimuli unknown at specification stage, one does not want to commit on any specific implementation; in particular, one does not want to state the behavior of *get(empty)*. More importantly, one wants to avoid constraining the behavior of *generate* as long as possible, most likely until the implementation stage.

The above suggests a conservative notion of behavioral equivalence, namely that two states are equivalent if and only if they appear to be the same to clients, under experiments that *they* can perform on the system. Since the rate and the policy for generating new numbers is intended to be beyond the capabilities of the client processes, this suggests that in this example, Γ, the set of behavioral operations, should only contain *get* and *remove*. This behavioral equivalence is also consistent with the view that, in case of a system crash, the numbers which are already generated are stored and therefore available at the next restart, while the numbers to be generated are totally unpredictable[7]. Let us now consider a plausible model A of the hidden specification above, namely one in which A_{State} contains pairs $[l, n]$, where l is a list of natural numbers (the generated numbers which were not yet consumed) and n is some counter, in which A_{get} returns the first element of l if it exists or 0 otherwise, in which A_{remove} lets n unchanged and removes the first element of l if it exists, and in which $A_{generate}([l, n])$ adds $f(n)$ to l and increments n, where $f : Nat \to Nat$ is some complicated function. Note then that two states $[l, n]$ and $[l', n']$ are behaviorally equivalent if and only if $l = l'$, and especially that $A_{generate}$ typically does *not* preserve this behavioral equivalence relation, because $f(n)$ may be different from $f(n')$ for different n, n'! If one wants to enforce a stack storage policy for the generated numbers, like in [27,29], then one can add the hidden equation

$$(\forall S : State)\ remove(generate(S)) = S.$$

Note that if this is the case then one can easily see that

$$(\forall S : State)\ generate(remove(generate(S))) = generate(S)$$

does not hold in A, so equational reasoning needs to be modified in order to be sound for behavioral reasoning.

Reuse. Another practical reason that motivated the introduction of behaviorally non-congruent operations came from the natural desire of reuse of specifications. Consider, e.g., a behavioral specification of lists of natural numbers

[7] The concept of "system crash" is of course hard and not intended to be formally captured as part of the hidden algebra models, but it is useful to motivate the conservative choice of behavioral operators and equivalence.

providing the usual operators *car* and *cdr*, and suppose that one wants to define a behavioral specification of sets on top of it. Since behavioral equality on sets, defined as "indistinguishability under membership", is different from the behavioral equality on lists, one simple and elegant way to define sets is to add to lists a membership operator $\_ \in \_$ together with the equation

$$(\forall L : List; N : Nat)\ N \in L = (N == car(L)) \vee (N \in cdr(L)),$$

and then to rename the sort *List* to *Set* and to set Γ to contain only the membership operation. This way, *car* and *cdr* will be correctly assumed to be non-congruent, so one cannot "prove" wrong facts such as $3 = car(3,4,5) = car(5,4,3) = 5$, just because the lists $(3,4,5)$ and $(5,4,3)$ are behavioral equivalent as sets.

One can argue that there may be other ways to avoid using operations which are not intended to preserve the behavioral equivalence, such as *car* and *cdr* above, for example by hiding them to the outside world or by wrapping the reused sort into another sort. In the former case, even though the non-congruent operators are not accessible from the outside, one still needs some criteria or techniques to ensure that the non-congruent operations are not indirectly used improperly *inside* the module defining the new behavior; for example, the property $3 = 5$ above was proved inside the module defining sets and, since it does not contain any forbidden operator, can be seen as a public property of sets. In the latter case, wrapping a hidden sort into another by introducing a new operator, e.g., $[\_] : List \to Set$, and then defining the behavioral equivalence on the new sort using any desired infrastructure on the old one, may look like a good solution. However, in our experience, such a solution leads to ugly and hard to read specifications in large examples, and often requires one to redeclare and redefine on the new sort operations already existing on the old sort.

We have found that by allowing behaviorally non-congruent operations as part of behavioral specifications, that is, by allowing models in which some operations are not required to preserve the behavioral equivalence, one can uniformly remove all the problems above and increase the definitional power of behavioral specifications. Proofs become less elegant, but in our view this should have a lower priority. Equational deduction and term rewriting have already been adapted to non-congruent operations [51]. Moreover, syntactic and proof theoretic criteria for proving behavioral congruence of operators have been devised by several groups of scientists [8,16,32,44,51,52]; some of these criteria have already implemented in BOBJ and greatly increase the power of behavioral equational reasoning: in many cases these criteria are able to show that all the operations are behaviorally congruent, thus making equational reasoning sound.

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 [15,14] (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 [29] and it essentially tuned a similar construction previously presented in [6] to our hidden logic framework. The construction in [29] was sufficiently good to relate behavioral specification to information hiding, but not good enough to explain how induction can be used to prove behavioral properties. 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 the present work, 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.

Intuitively, the idea is to add a new sort for each type of experiment, that is pair (hidden sort, visible sort), and then to add constructors for generating new experiments from previous ones by extending them *at the bottom*. Thus one starts with the attributes, which are the simplest experiments, and then keeps adding appropriate methods right above the special variable $\bullet$. This way one can generate any experiment. In addition to experiment constructors, operations are also needed to "apply" the experiments on hidden terms. All these operations give a high-order flavor to the specification generated by "unhiding" a hidden signature. However, notice that it is nothing but an ordinary many-sorted equational specification. The following definition unhides Γ

rather than Σ, because in what follows we will only unhide the subsignature Γ of behavioral operators.

Definition 24 *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*[8] *$\sigma : \overline{s}\ h \to v$ 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*[9] *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 25 Set (Continued). The unhiding specification of the specification 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 .
```

[8] 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. $h_k \in H$; for notational simplicity, we placed the hidden argument under consideration last.

[9] Same observation as in footnote 8.

```
  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 26 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
```

With the constructors `head : -> Stream->Nat` and `_[tail] : Stream->Nat -> Stream->Nat` for the new sort `Stream->Nat`, one can generate terms of the form `head[tail][tail][...][tail]` of sort `Stream->Nat`, which correspond to the appropriate experiments. When applied on terms of sort `Stream`, they evaluate to the expected terms corresponding to applying the real experiments. ■

4.2 Unhiding a Hidden Algebra

The usual hidden algebras need to be transformed into standard algebras in order to satisfy the unhiding specifications of their hidden signatures. Unhiding of a hidden algebra is performed by adding experiments to it. We first need to define experiments locally to a hidden algebra.

Definition 27 *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. We let $\mathbb{C}^A_\Gamma[\bullet : s]$ and $\mathbb{E}^A_\Gamma[\bullet : s]$ denote the sets of (Γ, A)-contexts and (Γ, A)-experiments, respectively.*

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 27 are instances of those in Definition 10, 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 28 *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 $v \in V$ and each $\gamma \in \mathbb{E}^A_{\Gamma,v}[\bullet : s]$.*

Proof: By Theorem 17, it suffices to show that the relation $\sim$, defined by $a \sim_h a'$ iff $A_\gamma(a) =_v A_\gamma(a')$ for each $v \in V$ and each $\gamma \in \mathbb{E}^A_{\Gamma,v}[\bullet : s]$, is the largest hidden Γ-congruence. Let $\sigma : h_1...h_k v_{k+1}...v_n \to s$ be any operation in Γ (for simplicity, we assume that its hidden arguments occur as the first k arguments), let $a_i, a'_i \in A_{h_i}$ such that $a_i \sim_{h_i} a'_i$ for $i = 1, ..., k$, let $d_i \in A_{v_i}$ for $i = k+1, ..., n$, and let $v \in V$ and $\gamma \in \mathbb{E}^A_{\Gamma,v}[\bullet : s]$ (if the sort s is visible, delete all occurrences of γ in the proof that follows and replace terms of the form $\gamma[t]$ by just t). Let γ_i be the term $\gamma[\sigma(a'_1, ..., a'_{i-1}, \bullet : h_i, a_{i+1}, ..., a_k, d_{k+1}, ..., d_n)]$ for each $i = 1, ..., k$. Since $a_i \sim_{h_i} a'_i$ one gets that $A_{\gamma_i}(a_i) = A_{\gamma_i}(a'_i)$. Letting a and a' denote the elements $A_\sigma(a_1, ..., a_k, d_{k+1}, ..., d_n)$ and $A_\sigma(a'_1, ..., a'_k, d_{k+1}, ..., d_n)$ respectively, notice that $A_\gamma(a) = A_{\gamma_1}(a_1)$, $A_{\gamma_i}(a'_i) = A_{\gamma_{i+1}}(a_{i+1})$ for $i = 1, ..., k-1$, and $A_{\gamma_k}(a'_k) = A_\gamma(a')$. Therefore, $A_\gamma(a) = A_\gamma(a')$. Since γ was chosen arbitrarily, we obtain that $a \sim_s a'$, i.e., $\sim$ is preserved by σ, and so $\sim$ is a hidden Γ-congruence. Because all operations in Γ preserve hidden Γ-congruences, so do the terms in $\mathbb{C}^A_\Gamma[\bullet : s]$. In particular, terms in $\mathbb{E}^A_\Gamma[\bullet : s]$ take congruent elements to identities. Therefore, any hidden Γ-congruence is included in $\sim$. □

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

Definition 29 *Given a hidden subsignature Γ of Σ and a hidden Σ-algebra A, let $\tilde{A}$ be the $(\Sigma \cup \tilde{\Gamma})$-algebra*[10] *defined by:*

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

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

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

The following proposition relates the unhiding of hidden signatures to the unhiding of corresponding hidden algebras. It essentially shows the expected result that the Γ-unhiding of a hidden Σ-algebra is a model of the unhiding specification of Γ, in the standard sense:

Proposition 30 *Given $\Gamma \subseteq \Sigma$ and a hidden Σ-algebra A, then $\tilde{A} \models_{\Sigma \cup \tilde{\Gamma}} E_\Gamma$.*

Proof: Let $(\forall Y; x : h)\ \sigma(Y)[x] = \sigma(Y, x)$ be an equation in E_Γ as in Definition 24, and let $\theta\colon Y \cup \{x\} \to \tilde{A}$ be a map. Then

$$\begin{aligned}
\theta(\sigma(Y)[x]) &= \tilde{A}_{_[_]}(\tilde{A}_\sigma(\theta(Y)), \theta(x)) \\
&= \tilde{A}_{_[_]}(\sigma(\theta(Y), \bullet), \theta(x)) \\
&= A_{\sigma(\theta(Y), \bullet)}(\theta(x)) \\
&= A_\sigma(\theta(Y), \theta(x)) \\
&= \theta(\sigma(Y, x)).
\end{aligned}$$

Now, let $(\forall Y; Exp : h' \to\!\!\triangleright\ v; x : h)\ Exp[\sigma(Y)][x] = Exp[\sigma(Y, x)]$ be an equation of the other type in E_Γ, and let $\theta\colon Y \cup \{Exp : h' \to\!\!\triangleright\ v; x : h\} \to \tilde{A}$ be any map. Then

$$\begin{aligned}
\theta(Exp[\sigma(Y)][x]) &= \tilde{A}_{_[_]}(\tilde{A}_{_[\sigma_]}(\theta(Exp), \theta(Y)), \theta(x)) \\
&= \tilde{A}_{_[_]}(\theta(Exp)(\sigma(\theta(Y), \bullet)), \theta(x)) \\
&= A_{\theta(Exp)(\sigma(\theta(Y), \bullet))}(\theta(x)) \\
&= A_{\theta(Exp)(\sigma(\theta(Y), \theta(x)))} \\
&= A_{\theta(Exp)}(A_\sigma(\theta(Y), \theta(x))) \\
&= \tilde{A}_{_[_]}(\theta(Exp), A_\sigma(\theta(Y), \theta(x))) \\
&= \theta(Exp[\sigma(Y, x)]).
\end{aligned}$$

Therefore, it follows that $\tilde{A} \models_{\Sigma \cup \tilde{\Gamma}} E_\Gamma$. □

The following proposition is very important because, by Proposition 2 via some further results presented in the sequel, it essentially allows one to soundly use inductive proofs on the newly added sorts due to unhiding:

Proposition 31 *$\tilde{A}$ is $(H \to\!\!\triangleright V, Z)$-reachable, for any $(H \cup V)$-indexed set Z.*

Proof: Let $(h \to\!\!\triangleright v) \in (H \to\!\!\triangleright V)$ and let $\gamma \in \mathbb{E}^A_{\Gamma,v}[\bullet : h]$. Pick an experiment ρ in $\mathbb{E}_{\Gamma,v}[\bullet : h]$ replacing each occurrence of an element of A in γ by a *distinct* variable in Z, and let $\theta\colon Z \to A$ assign to each such variable its correspondent element in A. Then it is clear that $\theta(\rho) = \gamma$. Therefore, $\tilde{A}$ is $(H \to\!\!\triangleright V, Z)$-reachable. □

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 19:

Definition 32 *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 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 $\tilde{\mathbb{B}} = (\Sigma \cup \tilde{\Gamma}, \tilde{E} \cup E_\Gamma)$ is* **the unhiding of** $\mathbb{B}$.

Notice that $\tilde{\mathbb{B}}$ is finite whenever $\mathbb{B}$ is finite.

Example 33 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 34 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 35 *Given a behavioral specification* $\mathbb{B} = (\Sigma, \Gamma, E)$*, a* Σ*-equation* e*, and a hidden* Σ*-algebra* A*, then*

(1) $A \models^{\Gamma}_{\Sigma} e$ *iff* $\tilde{A} \models_{\Sigma \cup \tilde{\Gamma}} \tilde{e}$*,*
(2) $A \models \mathbb{B}$ *iff* $\tilde{A} \models \tilde{\mathbb{B}}$*, and*
(3) $\tilde{\mathbb{B}} \models \tilde{e}$ *implies* $\mathbb{B} \models e$*.*

Proof: We treat both fixed-data and loose-data at the same time.

(1) Let e be the Σ-equation $(\forall X)\ t = t'$. If the sort of t, t' is visible, then the result is easy because of the satisfaction condition property of equational logics and because $\tilde{A}\restriction_{\Sigma} = A$. If the sort of t, t' is hidden, i.e., $\tilde{e}$ is the set of $(\Sigma \cup \tilde{\Gamma})$-equations $\{(\forall X; Exp : h \to\!\!\!\!\triangleright\ v)\ Exp[t] = Exp[t'] \mid v \in V\}$, then

$$
\begin{aligned}
A \models^{\Gamma}_{\Sigma} e \text{ iff } & \theta(t) \equiv^{\Gamma}_{\Sigma} \theta(t') \text{ for any } \theta\colon X \to A \\
& \text{iff } A_{\gamma}(\theta(t)) = A_{\gamma}(\theta(t')) \text{ for any } \gamma \in \mathbb{E}^{A}_{\Gamma}[\bullet : h] \text{ and any } \theta\colon X \to A \\
& \text{iff } \tilde{\theta}(t) = \tilde{\theta}(t') \text{ for any } v \in V \text{ and any } \tilde{\theta}\colon X \cup \{z : (h \to\!\!\!\!\triangleright\ v)\} \to A \\
& \text{iff } \tilde{A} \models_{\Sigma \cup \tilde{\Gamma}} \tilde{e}.
\end{aligned}
$$

(2) It follows by *1.* and Proposition 30.

(3) If $A \models \mathbb{B}$ then $\tilde{A} \models \tilde{\mathbb{B}}$ by *2.* above, therefore $\tilde{A} \models_{\Sigma \cup \tilde{\Gamma}} \tilde{e}$. Then it follows by *1.* above that $A \models^{\Gamma}_{\Sigma} e$. □

This proposition suggests that in order to show that e is a behavioral consequence of $\mathbb{B}$, it suffices to show that $\tilde{e}$ is an equational consequence of $\tilde{\mathbb{B}}$. As shown next, this simple proof technique is too weak in practical situations. Note that *(3)* cannot be an equivalence because it would otherwise provide a complete calculus for behavioral satisfaction, which is incomplete [10].

5 Practical Importance: Proving Behavioral Equivalence

If one wants to prove `SET` $\models$ (∀`S,S':Set`) `S U S' = S' U S` by *(3)* in Proposition 35, then one is stuck since one has to prove by ordinary equational rea-

soning `SET~` $\models$ (∀`S,S':Set; Exp:(Set->Bool)`) `Exp[S U S'] = Exp[S' U S]`, which is impossible. Some kind of induction on experiments is needed.

Definition 36 *Given a behavioral specification* $\mathbb{B} = (\Sigma, \Gamma, E)$ *and a* Σ*-equation* e*, then* $\tilde{\mathbb{B}}$ *is said to* $(H \twoheadrightarrow V, Z)$**-inductively satisfy** e*, which is written* $\tilde{\mathbb{B}} \models_{Ind(H \twoheadrightarrow V, Z)} \tilde{e}$*, 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 $\tilde{\mathbb{B}}$. 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 $\tilde{\mathbb{B}}$, 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 37 $\tilde{\mathbb{B}} \models_{Ind(H \twoheadrightarrow V, Z)} \tilde{e}$ *implies* $\mathbb{B} \models\!\models e$.

Proof: Let A be any hidden Σ-algebra such that $A \models\!\models \mathbb{B}$ and let e be the equation $(\forall X)\ t = t'$. By Proposition 35, $\tilde{A} \models \tilde{\mathbb{B}}$. We claim that $\tilde{A} \models_{\Sigma \cup \tilde{\Gamma}} \tilde{e}$. Indeed, let $\mathbb{P}^{\tilde{A}}$ be the $(H \twoheadrightarrow V)$-indexed predicate on $\tilde{A}\restriction_{(H \twoheadrightarrow V)}$ with $\mathbb{P}^{\tilde{A}}(\gamma)$ for some $\gamma \in \mathbb{E}^A_{\Gamma,v}[\bullet : h]$ if and only if $A_\gamma(\theta(t)) = A_\gamma(\theta(t'))$ for any map $\theta : X \to A$, and let $\mathbb{P}^T$ be the predicate on $(T_{\Sigma \cup \tilde{\Gamma}}(Z)/_{\tilde{E} \cup E_\Gamma})\restriction_{(H \twoheadrightarrow V)}$ with $\mathbb{P}^T(\rho)$ for some $\rho \in T_{\Sigma \cup \tilde{\Gamma}, (h \twoheadrightarrow v)}(Z)$ if and only if $\widehat{\rho[\alpha(t)]} = \widehat{\rho[\alpha(t')]}$ as equivalence classes in $T_{\Sigma \cup \tilde{\Gamma}}(Z)/_{\tilde{E} \cup E_\Gamma}$, where $\alpha : X \to Z$ is a map assigning to each variable in X a distinct variable in Z which is also distinct from those that occur in ρ. Notice that $\mathbb{P}^T = (T_{\Sigma \cup \tilde{\Gamma}}(Z)/_{\tilde{E} \cup E_\Gamma})\restriction_{(H \twoheadrightarrow V)}$. Then by Propositions 2 and 31, it suffices to show that $\mathbb{P}^T(\rho)$ implies $\mathbb{P}^{\tilde{A}}(\tau(\rho))$ for any $\tau : Z \to \tilde{A}$. Let us fix a map $\tau : Z \to \tilde{A}$, let $\theta : X \to A$ be any map, and let $\tau_\theta : Z \to \tilde{A}$ be such that $\tau_\theta\restriction_{\alpha(X)} = \theta$ and $\tau_\theta\restriction_{(Z - \alpha(X))} = \tau\restriction_{(Z - \alpha(X))}$. Since τ_θ extends to a morphism $T_{\Sigma \cup \tilde{\Gamma}}(Z)/_{\tilde{E} \cup E_\Gamma} \to \tilde{A}$, one gets $\tau_\theta(\rho[t]) = \tau_\theta(\rho[t'])$, that is, $\tilde{A}_{\tau(\rho)}(\theta(t)) = \tilde{A}_{\tau(\rho)}(\theta(t'))$. Hence $\tilde{A} \models_{\Sigma \cup \tilde{\Gamma}} \tilde{e}$, so by Proposition 35, $A \models\!\models^{\Gamma}_{\Sigma} e$. □

Proposition 37 suggests the following procedure to do behavioral proofs:

(1) Generate $\tilde{B}$ and $\tilde{e}$;
(2) Show $\tilde{\mathbb{B}} \models_{Ind(H \twoheadrightarrow V, Z)} \tilde{e}$ manually or using an inductive theorem prover;
(3) Conclude $\mathbb{B} \models\!\models e$.

We next analyze examples.

Example 38 Set (Continued). According to the proposition above, distributivity reduces to showing that `SET~` inductively satisfies (∀`S,S',S'':Set; Exp:Set->Bool`) `Exp[S&(S' U S'')] = 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
```

The above uses the theorem of constants and induction on experiments. Note that the experiments are trivial in this example, and so is the induction. ■

Example 39 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 [24,48].

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 $\mathbb{P}$ be the predicate on experiments such that $\mathbb{P}$(`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 $\mathbb{P}$(`Exp`) holds for all experiments `Exp` by structural induction. $\mathbb{P}$(`head`) holds because `head(s1)` is equal to `head(s1')` for any behaviorally equivalent `s1` and `s1'`. Assume $\mathbb{P}$(`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 $\mathbb{P}$(`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 [11] . We need only one in our proof:

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

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] .
```

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

```
***> should both be true
```

■

The inductive technique used in the examples above is called *context induction* [33] (see also [4] for related work). However, note that a major benefit of our translation is that any proof technique for the ordinary algebraic specification $\tilde{\mathbb{B}}$ is allowed, as far as it is sound at least for the models $\tilde{A}$ associated to hidden algebras A. 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 [48]. 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 40 *Given a behavioral specification $\mathbb{B} = (\Sigma, \Gamma, E)$ and a hidden Σ-algebra A, if $\Sigma \Box \tilde{\mathbb{B}}$ is the ordinary equational Σ-theory consisting of all the Σ-theorems of $\tilde{\mathbb{B}}$ then*

(1) $A \models\!\models \mathbb{B}$ iff $A \models \Sigma \Box \tilde{\mathbb{B}}$, and
(2) In the loose-data hidden algebra case, $\mathbb{B}$ and $\Sigma \Box \tilde{\mathbb{B}}$ have the same models.

Proof: *(1)* Since the sentences in $\Sigma \Box \tilde{\mathbb{B}}$ are $(\Sigma \cup \tilde{\Gamma})$-consequences of those in $\tilde{\mathbb{B}}$, the implication "$A \models\!\models \mathbb{B}$ implies $A \models \Sigma \Box \tilde{\mathbb{B}}$" follows immediately by *(2)* in Proposition 35 and the satisfaction condition of equational logics (because $\tilde{A}{\restriction_\Sigma} = A$). In order to prove the other implication, by Proposition 20, it suffices to show that the sentences in $\mathbb{E}_\Gamma[E]$ are among the Σ-sentences in $\Sigma \Box \tilde{\mathbb{B}}$. Let $\mathbb{E}_\Gamma[e]$ be an equation $(\forall X, var(\gamma))\ \gamma[t] = \gamma[t']$ in $\mathbb{E}_\Gamma(E)$, for some experiment γ in $\mathbb{E}_\Gamma[\bullet : h]$ and some equation $(\forall X)\ t = t'$, say e, in E. If γ has the form $\sigma_1(\overline{t_1}, \sigma_2(\overline{t_2}, \ldots, \sigma_k(\overline{t_k}, \bullet)\ldots))$ for some appropriate arguments (i.e., lists of terms) $\overline{t_1}, \overline{t_2}, \ldots, \overline{t_k}$, then let $\tilde{\gamma}$ be the term $\sigma_1(\overline{t_1})[\sigma_2(\overline{t_2})][\cdots][\sigma_k(\overline{t_k})]$ in $T_{\Sigma \cup \tilde{\Gamma}, (h \to\!\!\!\!\bullet\ v)}(var(\gamma))$. It can be relatively easily shown (using only the equations in E_Γ) by induction on the structure of γ, that $\tilde{\mathbb{B}} \models (\forall X, var(\gamma))\ \tilde{\gamma}[t] = \gamma[t]$ and $\tilde{\mathbb{B}} \models (\forall X, var(\gamma))\ \tilde{\gamma}[t'] = \gamma[t']$. Further, by instantiating the equation $(\forall X, z : (h \to\!\!\!\!\bullet\ v))\ z[t] = z[t']$, one gets that $\tilde{\mathbb{B}} \models (\forall X, var(\gamma))\ \tilde{\gamma}[t] = \tilde{\gamma}[t']$. Hence, $\tilde{\mathbb{B}} \models \mathbb{E}_\Gamma[e]$, i.e., $\mathbb{E}_\Gamma[e] \subseteq \Sigma \Box \tilde{\mathbb{B}}$. *(2)* follows immediately by *(1)*, noticing

that hidden algebras are nothing but algebras in the loose-data framework. □

Therefore, (finite) behavioral equational specifications can be translated into (finite) standard equational specifications in such a way that the two have the same models. This translation could also be formalized as some map between institutions [30], but this would go beyond the purpose of this paper.

7 Conclusion

By adding machinery for experiments, use it and then hide it, we showed how any behavioral Σ-specification $\mathbb{B}$ can be "unhidden" to an ordinary algebraic specification $\tilde{\mathbb{B}}$ over a larger signature, such that a model behaviorally satisfies $\mathbb{B}$ if and only if it satisfies, in the ordinary sense, the Σ-theorems of $\tilde{\mathbb{B}}$. The construction of $\tilde{\mathbb{B}}$ is algorithmic and finite when $\mathbb{B}$ 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 $\tilde{\mathbb{B}}$ to reason about behavioral equality in $\mathbb{B}$, despite the fact that neither of those is sound in $\mathbb{B}$.

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, and to potentially interesting and powerful modularization results. It would also be useful to extend the results in this paper to conditional equations whose conditions can be hidden; the technical problem here is that an infinite number of experiments is needed to test hidden conditions. Another direction of further research is to investigate to what extent the results presented in this paper can be applied to coalgebraic specification settings allowing types with sum codomains, e.g., to those proposing novel interesting notions of "coterms" and "coequations" such as Cîrstea [12], Kurz [36], Mossakowski *et al* [41], and Rothe *et el* [53] among many others.

References

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

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

[3] G. Bernot, M. Bidoit, and T. Knapik. Observational specifications and the indistinguishability assumption. *Theoretical Comp. Science*, 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*. Springer, 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 Formal Methods'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. Coalgebra semantics for hidden algebra: parameterized objects and inheritance. In *Recent Trends in Algebraic Development Techniques*, volume 1376 of *LNCS*. Springer, 1998.

[12] C. Cîrstea. A coequational approach to specifying behaviours. In *Proceedings of CMCS'99*, volume 19 of *ENTCS*. Elsevier, 1999.

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

[14] M. Clavel, F.J. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J.F. Quesada. Maude: Specification and Programming in Rewriting Logic. *Theoretical Computer Science*, 285:187–243, 2002.

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

[16] A. Corradini, R. Heckel, and U. Montanari. From SOS specifications to structured coalgebras: How to make bisimulation a congruence. In *Proceedings of CMCS'99*, volume 19 of *ENTCS*. Elsevier, 1999.

[17] R. Diaconescu and K. Futatsugi. *CafeOBJ Report*. World Scientific, 1998. AMAST Series in Computing, volume 6.

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

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

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

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

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

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

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

[25] 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.

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

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

[28] 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.

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

[30] J. Goguen and G. Roşu. Institution morphisms. *Formal Aspects of Computing*, 13(3-5):274–307, 2002.

[31] 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.

[32] J. F. Groote and F. W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.

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

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

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

[36] A. Kurz. Specifying coalgebras with modal logic. In *Proceedings of CMCS'98*, volume 11 of *ENTCS*. Elsevier, 1998.

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

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

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

[40] 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.

[41] T. Mossakowski, H. Reichel, M. Roggenbach, and L. Schroder. Algebraic-coalgebraic specification in CoCASL. In *Proceedings of WADT'02*, volume 2755 of *LNCS*, pages 376–392. Springer, 2002.

[42] P. Padawitz. Swinging data types: Syntax, semantics, and theory. In *Proceedings, WADT'95*, volume 1130 of *LNCS*, pages 409–435. Springer, 1996.

[43] 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.

[44] P. Padawitz. Swinging types = functions + relations + transition systems. *Theoretical Computer Science*, 243:93–165, 2000.

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

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

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

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

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

[50] G. Roşu. On implementing behavioral rewriting. In *Proceedings of the ACM SIGPLAN RULE'02*, pages 43–52. ACM Press, 2002.

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

[52] G. Roşu and J. Goguen. Circular coinduction. In *International Joint Conference on Automated Reasoning (IJCAR'01)*. 2001. Short paper. An extended version appeared as UCSD Technical Report CSE2000–0647.

[53] J. Rothe, B. Jacobs, and H. Tews. The coalgebraic class specification language CCSL. *Journal of Universal Computer Science*, 7:175–193, 2001.

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

[55] H. Tews. Coalgebras for binary methods. In *Proceedings of CMCS'00*, volume 33 of *ENTCS*. Elsevier, 2000.

[56] H. Tews. Coalgebras for binary methods: Properties of bisimulations and invariants. *Theoretical Informatics and Applications*, 35(1):83–111, 2001.

[57] H. Tews. Greatest bisimulations for binary methods. In *Proceedings of CMCS'02*, volume 65(1) of *ENTCS*. Elsevier, 2002.

[58] M. Walicki and S. Meldal. Algebraic approaches to nondeterminism: An overview. *ACM Computing Surveys*, 29:30–81, 1997.