Towards Behavioral Maude: Behavioral Membership Equational Logic

José Meseguer¹ and Grigore Roșu²

¹Department of Computer Science
University of Illinois at Urbana-Champaign
²NASA Ames Research Center - RIACS

Abstract

How can algebraic and coalgebraic specifications be integrated? How can behavioral equivalence be addressed in an algebraic specification language? The hidden-sorted approach, originating in work of Goguen and Meseguer in the early 80’s, and further developed into the hidden-sorted logic approach by researchers at Oxford, UC San Diego, and Kanazawa offers some attractive answers, and has been implemented in both BOBJ and CafeOBJ. In this work we investigate both further extensions of hidden logic, and an extension of the Maude specification language called BMaude supporting this extended hidden-sorted semantics.

Maude’s underlying equational logic, membership equational logic, generalizes and increases the expressive power of many-sorted and order-sorted equational logics. We develop a hidden-sorted extension of membership equational logic, and give conditions under which theories have both an algebraic and a coalgebraic semantics, including final (co-)algebras. We also discuss the language design of BMaude, based on such an extended logic and using categorical notions in and across the different institutions involved. We also explain how Maude’s reflective semantics provides a systematic method to extend Maude to BMaude within Maude, including module composition operations, evaluation, and automated proof methods.

Key words: Membership and hidden algebra, coalgebra, Maude.

1 Introduction

As the title suggests, this is work in progress. The general aim is the semantic integration of different notions of behavioral equivalence emerging from different perspectives, such as hidden-sorted logics, coalgebras, and concurrency theory. Based on such an integration we then wish to explore specification languages supporting behavioral notions, and integrating several views or paradigms, such as the algebraic, coalgebraic, object-oriented, and rewriting logic views; we use BMaude as a specification language design helping us advance our ideas. Advances in semantic integration at the theoretical and
specification language levels should then make possible a semantic integration and combination of different formal methods, including inductive, coinductive, equational, rewriting logic, and temporal and modal logic methods; and also of lighter methods such as runtime verification, and specification-based testing and monitoring. In this way, it should be possible to deal more effectively with the different aspects of software systems; to design, implement and maintain them better; and to gain greater assurance about their correct behavior.

The notion of behavioral equivalence has had an interesting development. Some of the intuitions go back more than fifty years to automata theory [72], but the elegant idea of formalizing behavioral equivalence as “indistinguishability under experiments” via the notion of context seems to have been introduced by Reichel [81] (see also [82]) in 1981 in the context of partial algebras, and since then has been used and adapted by many researchers in various settings, including us. Among other early references to behavioral satisfaction and equivalence, we would mention the 1982 work by Goguen and Meseguer [49] on “abstract modules” that also presents a final algebra theorem for behavioral equivalence. Hidden logics give the behavioral equivalence a central role, and they originate in a paper by Goguen [40] in 1989, under the name of hidden algebra. The first systematic exposition of hidden algebra was presented by Goguen and Diaconescu in [33] in 1994, and then more deeply investigated by Goguen and Malcolm [48,47,66,46]. Order-sorted versions of hidden algebra were explored in [66,46,18,14]. Motivated by the continuously increasing number of new challenging practical situations we and other scientists encountered, hidden logics have undergone a series of generalizations and notational simplifications in works by Goguen, Roșu and Lin [91,51,86,93,45], showing that many results previously thought to depend on the monadic nature of operations in hidden sorts hold in a more general setting, with polyadic operations, non-behavioral operations, and even with no fixed-data universe. These results lead us to the development of BOBJ [45], an executable behavioral specification language in the OBJ family. A comprehensive exposition of various hidden logics together with historical background and plenty of examples can be found in Roșu [88]. Important contributions to hidden logics were also made by Diaconescu, Futatsugi and others within the CafeOBJ project at JAIST in Japan [28,30,29,60,73] under the name coherent hidden algebra, by Bernot, Bidoit, Hennicker and others [2,56,3,10,7] that lead to the evolution of observational logic [57], and by Padawitz on swinging types [77].

The important notion of behavioral equivalence in hidden membership algebra is closely related to its counterpart in coalgebra, namely bisimulation (see also Malcolm and Ciurstea [65]). The full relationship between hidden logics and coalgebra is still to be explored, but our understanding at this moment is that the models of a special segment of behavioral membership equational logic, called coalgebraic (fixed-data) behavioral membership equational logic (presented in detail in Section 4), are isomorphic to the coalgebras over certain “algebraic” functors. A good introduction to coalgebra and coal-
gebraic coinduction, as well as to the duality between algebra and coalgebra, is the work by Jacobs and Rutten [63,94]. There is an increasing recent interest in combining coalgebra with algebra. For example, Hensel and Reichel [59] use it to define equations on terminal coalgebras, and Hennicker, Kurz and Bidoit [58,61] to extend current coalgebraic techniques with observational logic principles. Both algebraic and coalgebraic settings inherently live together in our approach, without any special treatment for either of them: if one wants “coalgebraic” features then one should just add hidden sorts, think of terms of hidden sort as “states,” and then use coinduction and behavioral equational deduction to prove behavioral properties about them, or induction and standard equational reasoning to prove properties about the visible world, the so called “data”; hidden constants and non-monadic operations in hidden sorts do not bring any trouble, so one has no such limitations. This is our simple methodological approach to behavioral specification and verification, and is faithfully supported by BOBJ [45,88] and the design of BMaude.

The works closest to our aims fall within the OBJ family of specification languages, to which Maude also belongs, and includes work on both BOBJ [45,88] and CafeOBJ [28,31]. Building on the research contributions made by those two languages, we seek to address two main open problems:

(1) the development of behavioral equational logic in increasingly more expressive equational frameworks, such as total and partial membership equational logics [69,12,71];

(2) a fuller understanding of the behavioral aspects of rewriting logic, including a clarification of how the object-oriented notions supported by behavioral equational logic and by rewriting logic fit together [68,31].

In this paper—after giving some preliminaries on hidden many-sorted equational logic and on membership equational logic in Section 2—we focus our efforts on problem (1). Given that rewriting logic, at least in its Maude version, contains membership equational logic as a sublogic, solving (1) is both a foundation and a necessary prerequisite for solving problem (2).

It is well-known that membership equational logic (MEL) is a very expressive logical framework for algebraic specification, in the sense that many other formalisms, both total and partial, and having various type disciplines such as, for example, order-sorted or equational type logics can be naturally represented as special cases [69]. Furthermore, under reasonable confluence and termination assumptions [12], membership equational logic can be efficiently executed by rewriting, as demonstrated by its Maude implementation.

It seems therefore useful to investigate how membership equational logic can be generalized to behavioral membership equational logic (BMEL) and to explore how the extra expressive power can be exploited at the behavioral level. We do this in Section 3 where we also explain how BMEL is an institution, and furthermore a logic, containing MEL as a sublogic. Then, in Section 4 we study sufficient conditions on a BMEL-specification for its
models to form a category of coalgebras and to have a final coalgebra, thus intimately relating the algebraic and coalgebraic views. Building on these categorical and institutional foundations, in Section 5 we then explore and illustrate with examples the language design of BMaude for its MEL and BMEL facets, including its (MEL) functional theories and modules (which may be parameterized and can have freeness constraints); and its (BMEL) functional behavioral theories and modules (that can also be parameterized, and can have freeness and/or finality constraints). In Section 6 we explain how, using Maude’s reflective capabilities, it is possible to develop a BMaude implementation written entirely in Maude, without any need to extend or modify the underlying Maude C++ implementation. In particular, reflection allows natural and relatively easy implementations of behavioral rewriting, and of coinductive rewriting, as explained in Section 7. As work in progress, there are of course many issues not yet resolved, such as: solving problem (2), the integration of different formal methods, and the development of case studies and applications. In the concluding remarks (Section 8) we include a discussion of some of those issues, and our plans and tentative ideas on how to address them in future work. The proofs of the results in this paper are omitted, but they will appear elsewhere soon.

We warmly thank Narciso Martí-Oliet for his very detailed comments on previous drafts of this paper, and also to all colleagues that contributed to the development of BOBJ, CafeOBJ, Maude, as well as to the development of hidden and membership algebra.

2 Preliminaries

We assume the reader familiar with basic equational logic and algebraic notions. Given an $S$-sorted signature $\Sigma$ and an $S$-indexed set of variables $Z$, we let $T_\Sigma(Z)$ denote the $\Sigma$-term algebra over variables in $Z$. If $V \subseteq S$ then $\Sigma|_V$ is a $V$-sorted signature consisting of all those operations in $\Sigma$ defined entirely with sorts in $V$, that is, $\Sigma|_V = \{ \sigma : w \to s \in \Sigma \mid w \in V^*, s \in V \}$. We often let $\sigma(X)$ denote the term $\sigma(x_1,...,x_n)$ when the number of arguments of $\sigma$ and their order and sorts are not important. If only one argument is important, then to simplify writing we place it at the end; for example, $\sigma(X,t)$ is a term having $\sigma$ as root with only variables as arguments except one, and we do not care which one, which is $t$. $\text{Der}(\Sigma)$ is the derived signature of $\Sigma$, which basically contains all the $\Sigma$-terms viewed as operations. If $t$ is a $\Sigma$-term and $A$ is a $\Sigma$-algebra, then $A_t : A^{\text{var}(t)} \to A$ is the interpretation of $t$ in $A$: for any map $\theta : \text{var}(t) \to A$, $A_t(\theta)$ is the evaluation of $t$ in $A$ where all the variables in $t$ are replaced by their concrete values given by $\theta$; this if often denoted just $\theta(t)$ for simplicity. If a variable of $t$, say $\star$, is of special importance, then we can view the evaluation of $t$ in two steps, that is, $A_t : A \to (A^{(\text{var}(t)-\{\star\})} \to A)$ with the obvious meaning. If $f : A \to B$ and $g : B \to C$ are functions/morphisms then we let $f;g : A \to C$ denote their composition, so we use diagrammatic order.
2.1 Institutions

Institutions were introduced by Goguen and Burstall [41,42] to formalize the concept of logical system.

**Definition 2.1** An institution \( \mathcal{I} = (\text{Sign}, \text{Mod}, \text{Sen}, \models) \) is a tuple consisting of a category \( \text{Sign} \) whose objects are called signatures, a functor \( \text{Mod} : \text{Sign} \to \text{Cat}^{op} \) giving for each signature \( \Sigma \) a category of \( \Sigma \)-models, a functor \( \text{Sen} : \text{Sign} \to \text{Set} \) giving for each signature a set of \( \Sigma \)-sentences, and a \( |\text{Sign}| \)-indexed relation \( \models = \{ \models \}_{\Sigma \in |\text{Sign}|} \) with \( \models \subseteq |\text{Mod}(\Sigma)| \times |\text{Sen}(\Sigma)| \), such that for any signature morphism \( \phi : \Sigma \to \Sigma' \), the following diagram

\[
\begin{array}{cc}
\Sigma & |\text{Mod}(\Sigma)| \\
\phi \downarrow & \models \downarrow & \phi \downarrow \\
\Sigma' & |\text{Mod}(\Sigma')| \\
\end{array}
\]

commutes, that is, the following satisfaction condition

\[ M' \models_{\Sigma'} \text{Sen}(\phi)(f) \iff \text{Mod}(\phi)(M') \models_{\Sigma} f \]

holds for all \( M' \in |\text{Mod}(\Sigma')| \) and \( f \in |\text{Sen}(\Sigma)| \).

We often write only \( \phi \) instead of \( \text{Sen}(\phi) \) and \( \models_{\phi} \) instead of \( \text{Mod}(\phi) \); the functor \( \models_{\phi} \) is called the reduct functor associated to \( \phi \). With this notation, the satisfaction condition becomes

\[ M' \models_{\Sigma'} \phi(f) \iff M' \models_{\phi} \models_{\Sigma} f . \]

We also use the satisfaction notation with a set of sentences \( F \) on its right side, letting \( M \models_{\Sigma} F \) mean that \( M \) satisfies each sentence in \( F \), and further extend this notation by letting \( F \models_{\Sigma} F' \) mean that \( M \models_{\Sigma} F' \) for any \( \Sigma \)-model \( M \) with \( M \models_{\Sigma} F \). We may omit the subscript \( \Sigma \) in \( \models \) when it can be inferred from context. The closure of a set of \( \Sigma \)-sentences \( F \), denoted \( F^\bullet \), is the set of all \( f \) in \( \text{Sen}(\Sigma) \) such that \( F \models_{\Sigma} f \). The sentences in \( F^\bullet \) are often called the theorems of \( F \). Closure is obviously a closure operator, that is, it is extensive, monotonic and idempotent, i.e., \( F \subseteq F^\bullet \), \( F \subseteq F' \) implies \( F^\bullet \subseteq F'^\bullet \), and \( F'^{\bullet \bullet} = F'^\bullet \). A specification (also called theory in this paper) is a pair \( (\Sigma, F) \) with \( F \) a set of \( \Sigma \)-sentences, and a morphism of theories \( \phi : (\Sigma_1, F_1) \to (\Sigma_2, F_2) \) is a morphism of signatures \( \phi : \Sigma_1 \to \Sigma_2 \) such that \( \phi(F_1) \subseteq F_2^\bullet \). We let \( \text{Th}_{\mathcal{I}} \) denote the category of theories of an institution \( \mathcal{I} \).

2.2 Hidden Many-Sorted Logic

Hidden algebra was formally introduced by Goguen in [40] to give algebraic semantics for the object paradigm, developed further in [43,46,47,48] among many other places, but it has its roots in earlier works, such as that by Reichel [81] and those by Goguen and Meseguer [49,70]. One distinctive feature
is a split of sorts into visible and hidden, where visible sorts are for data and hidden sorts are for objects and states. A model, or hidden algebra, is an abstraction of an implementation, consisting of the possible states, with concrete functions for the operations. Hidden logic (HL) \cite{88} is the generic name for various logics closely related to hidden algebra, giving sound rules for behavioral reasoning that are easily automated. Following \cite{15}, we distinguish two classes of hidden logics, depending on whether the data universe, of “builtins,” is assumed fixed or not. The first versions of hidden logic took the fixed-data approach, but we recently noticed that all our inference rules are sound for the larger class of models which need not protect data. Since there are also loose-data versions of hidden logics, such as coherent hidden algebra \cite{28,30} and observational logic \cite{4,6,57}, we decided not to restrict our exposition to the fixed-data case. Nevertheless, the fixed-data hidden logics are often desirable, since real applications use standard data-types such as booleans and integers rather than arbitrary models; for example, the alternating bit protocol cannot be proved correct unless implementations which do not distinguish 0 from 1 are forbidden. To the best of our knowledge, there currently are three systems that support and execute more or less general hidden logic specifications, namely CafeOBJ \cite{28}, Spike \cite{13}, and BOBJ \cite{45}.

A detailed presentation of various hidden logics appears in \cite{88} together with relations to many other concepts, a history of hidden algebra with citations, and proofs of some results mentioned but not proved here. We now introduce some of the most basic concepts, assuming familiarity with ordinary many-sorted algebra:

**Definition 2.2** Given disjoint sets $V, H$ called the sets of visible and hidden sorts, a loose-data hidden $(V,H)$-signature is a many-sorted $(V \cup H)$-signature $\Sigma$. A fixed-data hidden $(V,H)$-signature is a pair $(\Sigma,D)$ where $\Sigma$ is a loose-data hidden $(V,H)$-signature and $D$, called the data algebra, is a many-sorted $\Sigma|\downarrow V$-algebra. A loose-data hidden subsignature of $\Sigma$ is a loose-data hidden $(V,H)$-signature $\Gamma$ with $\Gamma \subseteq \Sigma$ and $\Gamma|\downarrow V = \Sigma|\downarrow V$. A fixed-data hidden subsignature of $(\Sigma,D)$ is a fixed-data hidden $(V,H)$-signature $(\Gamma,D)$ over the same data with $\Gamma \subseteq \Sigma$ a loose-data hidden subsignature. The operations in $\Sigma$ 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 (zero or more) visible arguments and hidden result are called hidden constants.

We may write “hidden signature” instead of “loose-data hidden $(V,H)$-signature” or “fixed-data hidden $(V,H)$-signature,” and $\Sigma$ instead of $(\Sigma,D)$.

**Definition 2.3** A loose-data hidden $\Sigma$-algebra $A$ is a $\Sigma$-algebra. A fixed-data hidden $(\Sigma,D)$-algebra $A$ is a $\Sigma$-algebra $A$ such that $A|_{\Sigma|\downarrow V} = D$. A loose-data $\Sigma$-morphism of hidden algebras is any $\Sigma$-morphism, while a fixed-data $(\Sigma,D)$-morphism is one which is the identity on the visible sorts.
We let $\text{HAlg}_\Sigma$ and $\text{HAlg}_{(\Sigma,D)}$ denote the categories of loose-data and fixed-data hidden algebras, respectively.

Again, we often write just “hidden algebra.” A hidden algebra can be regarded as a “blackbox,” the inside of which is not seen, since one is only concerned with its behavior under experiments. Notice that fixed-data hidden algebras protect their visible data; for example, an implementation of a stack of natural numbers does not corrupt its built-in natural numbers.

We next formalize the notion of “experiment,” which informally is an observation of an attribute of a system after it has been side-effected by some methods, using the mathematical concept of context; the symbol $\star$ below is a placeholder for the state being experimented upon. The interesting contexts are those for hidden sorts, but those for visible sorts are also allowed just to smooth the presentation.

**Definition 2.4** Given a hidden subsignature $\Gamma$ of $\Sigma$, an (appropriate) $\Gamma$-context for sort $s$ is a term in $T_\Gamma(\{\star : s\} \cup Z)$ having exactly one occurrence of a special variable $\star$ of sort $s$, where $Z$ is an infinite set of special variables. Let $C_\Gamma[\star : s]$ denote the set of all $\Gamma$-contexts for sort $s$, and let $\text{var}(c)$ denote the finite set of variables in a context $c$ except $\star$. Given a context $c \in C_\Gamma[\star : s]$, the sort $s'$ of $c$ as a term in $T_\Gamma(\{\star : s\} \cup Z)$ is called its **result sort**. A $\Gamma$-context with visible result sort is called a $\Gamma$-**experiment**; let $E_\Gamma[\star : s]$ denote the set of all $\Gamma$-experiments for sort $s$. When the sort of experiments is important, we use the notation $C_{\Gamma,s'}[\star : s]$ for the $\Gamma$-contexts of sort $s' \in V \cup H$ in $C_\Gamma[\star : s]$, while $E_{\Gamma,v}[\star : s]$ denotes all the $\Gamma$-experiments of sort $v \in V$ in $E_\Gamma[\star : s]$. If $c \in C_{\Gamma,s'}[\star : s]$ and $t \in T_{\Sigma,s}(X)$, then $c[t]$ denotes the term in $T_{\Sigma,s'}(\text{var}(c) \cup X)$ obtained from $c$ by substituting $t$ for $\star$; formally, $c[t] = (\star \rightarrow t)(c)$, where $(\star \rightarrow t) : T_\Sigma(\text{var}(c) \cup \{\star : s\}) \rightarrow T_\Sigma(\text{var}(c) \cup X)$ is the identity on $\text{var}(c)$ and takes $\star : s$ to $t$. Furthermore, $c$ generates a map $A_c : A_s \rightarrow [A_{\text{var}(c)} \rightarrow A_{s'}]$ on each $\Sigma$-algebra $A$, defined by $A_c(a)(\theta) = a_{\theta}(c)$, where $a_{\theta}$ takes $\star$ to $a$ and each $z \in \text{var}(c)$ to $\theta(z)$.

We now define a distinctive feature of hidden logic, behavioral equivalence. Intuitively, two states are behaviorally equivalent if and only if they cannot be distinguished by any experiment that can be performed on the system.

**Definition 2.5** Given a hidden $\Sigma$-algebra $A$ and a hidden subsignature $\Gamma$ of $\Sigma$, we define for all sorts $s \in V \cup H$ an equivalence relation between elements $a, a' \in A_s$ by $a \equiv_\Sigma^\Gamma a'$ iff $A_c(a)(\theta) = A_c(a')(\theta)$ for all $\Gamma$-experiments $c$ and all $(V \cup H)$-sorted maps $\theta : \text{var}(c) \rightarrow A$; we call this relation $\Gamma$-behavioral equivalence on $A$. We may write $\equiv$ instead of $\equiv_\Sigma^\Gamma$ when $\Sigma$ and $\Gamma$ can be inferred from context, and we write $\equiv_\Sigma$ when $\Sigma = \Gamma$. Given any $(V \cup H)$-equivalence $\sim$ on $A$, an operation $\sigma$ in $\Sigma_{s_1...s_n}$ 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 $\sigma$ is $\Gamma$-behaviorally congruent for $A$ iff it is congruent for $\equiv_\Sigma^\Gamma$. We often

---

1 Special variables are assumed different from any other variables in a given situation.
A hidden \(\Gamma\)-congruence on \(A\) is a \((V \cup H)\)-equivalence on \(A\) which is the identity on visible sorts and for which each operation in \(\Gamma\) is congruent.

Notice that behavioral equivalence is the identity on visible sorts, because contexts \(* : v\) are proper experiments in \(E_{\Gamma,v}[* : v]\) for all \(v \in V\).

The following is the basis for several important results, generalizing a result in [48] to operations that have more than one hidden argument or are not behavioral; see [91, 88] for a proof. Since final algebras do not necessarily exist in this setting, existence of a largest hidden \(\Gamma\)-congruence does not depend on them, as it does in coalgebra [94, 63, 61].

**Theorem 2.6** Given a hidden subsignature \(\Gamma\) of \(\Sigma\) and a hidden \(\Sigma\)-algebra \(A\), then \(\Gamma\)-behavioral equivalence is the largest hidden \(\Gamma\)-congruence on \(A\).

**Definition 2.7** A hidden \(\Sigma\)-algebra \(A\) \(\Gamma\)-behaviorally satisfies a \(\Sigma\)-equation \((\forall X) t = t'\) if \(t_1 = t'_1, ..., t_n = t'_n\), say \(e\), iff for each \(\theta : X \to A\), if \(\theta(t_i) \equiv \Gamma \theta(t'_i)\) for all \(1 \leq i \leq n\), then \(\theta(t) \equiv \Gamma \theta(t')\); in this case we write \(A \equiv F\). If \(F\) is a set of \(\Sigma\)-equations we then write \(A \equiv F\) whenever \(A\) \(\Gamma\)-behaviorally satisfies each \(\Sigma\)-equation in \(F\).

When \(\Sigma\) and \(\Gamma\) are clear from context, we may write \(\equiv\) instead of \(\equiv F\).

**Definition 2.8** A behavioral (or hidden) \(\Sigma\)-specification (or -theory) is a triple \((\Sigma, \Gamma, F)\) where \(\Sigma\) is a hidden signature, \(\Gamma\) is a hidden subsignature of \(\Sigma\), and \(F\) is a set of \(\Sigma\)-equations. The operations in \(\Gamma - \Sigma\) are called behavioral. We usually let \(B, B', B_1\), etc., denote behavioral specifications. A hidden \(\Sigma\)-algebra \(A\) \(\Gamma\)-behaviorally satisfies (or is a model of) a behavioral specification \(B = (\Sigma, \Gamma, F)\) iff \(A \equiv F\), and in this case we write \(A \equiv B\); we write \(B \equiv e\) if \(A \equiv B\) implies \(A \equiv e\). An operation \(\sigma \in \Sigma\) is behaviorally congruent for \(B\) iff \(\sigma\) is \(\Gamma\)-behaviorally congruent for each \(A\) such that \(A \equiv B\).

Many examples of non \(\Gamma\)-behaviorally congruent operations can be encountered in the context of programming languages [55]. For example, two programs in a given programming language can be considered equivalent if and only if they both terminate and return the same output (appropriate operations in \(\Gamma\) can be defined to enforce this natural relation of behavioral equivalence, like in [55]). However, in order to properly define the syntax and the semantics of a programming language, its behavioral specification needs to define various operations which do not preserve this behavioral equivalence, such as the length of a program, its running time, or its execution environment (two programs may declare a variable \(x\), one instantiate it to 0 and the other to 1, and then never use that variable).

Hidden logic can be organized as an institution in two interesting ways [51]. In one institution, the behavioral/congruent operations in \(\Gamma\) are declared as part of the signature, so a signature is actually a pair \((\Omega, \Gamma)\), and

\(^2\) A similar notion was given by Padawitz in [78].
the notions of sentence, model and satisfaction are defined as expected. Our institution for hidden membership algebra described later in the paper follows this pattern. In the other institution, the property of an operation to be behavioral/congruent is regarded as a semantical statement, so these operations are declared as sentences; however, the models then have to come with a builtin behavioral equivalence relation. It is shown in [51] that both institutions make sense and that there is actually an interesting institution map relating the two, called a “theoroidal forward morphism” in [52].

**Proposition 2.9** If \( B = (\Sigma, \Gamma, F) \) is a behavioral specification, then all operations in \( \Gamma \), and all hidden constants, are behaviorally congruent for \( B \).

Of course, depending on \( F \), other operations may also be congruent. [91] shows how congruence of operations can be proved, and also gives a first simple criterion to check congruence, which has been further generalized in [8,92]. As shown in [91], congruent operations can be added to or removed from \( \Gamma \) at our discretion when the equations in \( F \) do not have equalities of hidden sort in their conditions, which is usually the case. The following hidden equational deduction system is sound for behavioral satisfaction; the inference rules further illustrate why congruence of operations is important.

Let \( B = (\Sigma, \Gamma, F) \) be a behavioral specification and let us consider the following inference rules:

1. **Reflexivity**:  
   \[
   F \vdash_{\Sigma} (\forall X) \ t = t 
   \]

2. **Symmetry**:  
   \[
   F \vdash_{\Sigma} (\forall X) t = t' 
   
   F \vdash_{\Sigma} (\forall X) t' = t 
   \]

3. **Transitivity**:  
   \[
   F \vdash_{\Sigma} (\forall X) t = t' \quad F \vdash_{\Sigma} (\forall X) t' = t'' 
   
   F \vdash_{\Sigma} (\forall X) t = t'' 
   \]

   \[
   \begin{cases}
   a) & F \vdash_{\Sigma} (\forall X) t = t', \ t, t' \in T_{\Sigma, V}, v \in V \\
   & F \vdash_{\Sigma} (\forall X, W) \sigma(W, t) = \sigma(W, t'), \text{ for each } \sigma \in \text{Der}(\Sigma) \\
   \end{cases}
   \]

   \[
   \begin{cases}
   b) & F \vdash_{\Sigma} (\forall X) t = t', \ t, t' \in T_{\Sigma, H}, h \in H \\
   & F \vdash_{\Sigma} (\forall X, W) \delta(W, t) = \delta(W, t'), \text{ for each congruent } \delta \in \Sigma \\
   \end{cases}
   \]

4. **Congruence**:  
   \[
   F \vdash_{\Sigma} (\forall Y) t = t' \text{ if } t_1 = t'_1, ..., t_n = t'_n \text{ in } F \text{ and } \theta : Y \rightarrow T{\Sigma}(X) \\
   F \vdash_{\Sigma} (\forall X) \theta(t_1) = \theta(t'_1), ..., F \vdash_{\Sigma} (\forall X) \theta(t_n) = \theta(t'_n) 
   \]

5. **Modus Ponens**:  
   \[
   F \vdash_{\Sigma} (\forall X) \theta(t) = \theta(t') 
   \]

Note that, although the rules (1)-(3), and (5) are identical to those of many-sorted equational deduction, ordinary equational deduction is unsound.
for behavioral satisfaction, because the usual congruence deduction rule is unsound for operations that are not congruent. The rules above modify the congruence rule to account for this.

Unlike equational logics, the deduction system above is not complete. In fact, behavioral satisfaction is a \( \Pi_2^0 \)-hard problem \[15\] in general, even for unconditional equations, so there is no automatic procedure that can prove all true statements or disprove all false statements. Context induction \[56,39,5\] and hidden coinduction \[47,48\] are among the most popular proof techniques for behavioral equivalence in hidden logics, but unfortunately they both require intensive human intervention. Context induction is a mathematically elegant method based on well-founded induction on contexts. Despite efforts to reduce the number of useful contexts, it turns out to be pretty awkward to apply this method in practice, because it is often the case that the user has to provide non-trivial lemmas; as perhaps best shown in \[51\], where an OBJ compiler is shown behaviorally correct, context induction is almost impossible to control for large specifications. Also, proofs by coinduction, like proofs by bisimulation, require an appropriate binary relation to be provided in order to apply Theorem 2.6. It is, of course, inconvenient for the user to have to provide such a relation for any proof, so one would like to automate the process of finding a good relation as much as possible. Some interesting heuristics are suggested in \[25\], but as far as we know, user-defined lemmas are also often needed. A surprisingly powerful and fully automatic method, called circular coinductive rewriting \[45,88\], exploits the fact that most behavioral specifications are circular (or rather corecursive). BOBJ implements circular coinductive rewriting as well as a powerful congruence criterion, and can prove automatically most of the behavioral properties that we have encountered in the literature related to hidden logics, such as the following.

**Example 2.10** Infinite streams are common in the formal specification and verification of protocols, where they serve as inputs and outputs. The following is an executable specification of streams in BOBJ.

```plaintext
bth STREAM is sort Stream .
  protecting NAT .
  op head : Stream -> Nat .
  op tail : Stream -> Stream .
  op odd : Stream -> Stream .
  op even : Stream -> Stream .
  op zip : Stream Stream -> Stream .

  var N : Nat . vars S S’ : Stream .
  eq head(odd(S)) = head(S) .
  eq tail(odd(S)) = even(tail(S)) .
  eq head(even(S)) = head(tail(S)) .
  eq tail(even(S)) = even(tail(tail(S))) .
  eq head(zip(S,S’)) = head(S) .
  eq tail(zip(S,S’)) = zip(S’,tail(S)) .
end
```
As usual, head and tail give the stream’s first element and the rest of the stream, respectively, while odd and even give the streams of elements in the odd and even positions, respectively, and zip interleaves two streams.

A behavioral theory is declared in BOBJ via the keywords bth ... end, with the signature and the equations in between. All sorts declared in a behavioral theory are considered hidden; the visible sorts (here nat) are imported from some visible (data) specification (here NAT). Operations are behavioral by default; an operation not intended to be behavioral is given the attribute ncong. The models of a behavioral theory are the hidden algebras that behaviorally satisfy all its equations. In our case, the standard model is that of infinite lists of natural numbers, with head and tail as expected (the tail of an infinite list is infinite). For example, odd(1 2 3 4 5 6 7 8 9 ...) is 1 3 5 7 9 ..., even(1 2 3 4 5 6 7 8 9 ...) is 2 4 6 8 ..., and zip(1 3 5 7 9 ..., 2 4 6 8 ...) is 1 2 3 4 5 6 7 8 9 .... However, there may also be non-standard models; for example, the model with exactly one element in each carrier is valid for any loose-data hidden theory, but such a model is excluded in the above example because of the protecting NAT importation, which forces the meaning of NAT to be fixed.

In this example, Γ contains all the operations, because all of them are behavioral by default. Therefore, head(⋆), tail(⋆), head(tail(zip(odd(⋆), z))), are all Γ-contexts. If A is the standard infinite list model, then two lists are behaviorally equivalent if and only if they have the same elements in the same order. However, there are models of the above specification where a stream is an infinite tree, or some other infinite structure, so that elements can be behaviorally equivalent but not equal.

One can show that head and tail suffice as behavioral operations, since together they can observe all behaviors of states and thus correctly define the behavioral equivalence, i.e., they form a cobasis [91,88], while the other operations are all behaviorally congruent. BOBJ, due to its congruence criterion, can automatically find that {head, tail} is indeed a cobasis, and also that all the other operations are congruent. Once a cobasis is found, one can start the circular coinductive rewriting engine via the command cred. For example,

\[ \text{cred zip(odd(S), even(S)) == S} \]
\[ \text{cred zip(odd(S), even(odd(S))) == S} \]

As expected, the first reduction returns true, while the second returns false. Notice that these proofs are nontrivial. A proof of the first behavioral equality using context induction would require four user-defined nontrivial lemmas, each proved also by context induction. In fact, we are not aware of any other behavioral proof engine that can show these properties automatically.

2.3 Membership Algebra

In this section we recall some (total) membership equational logic (MEL) definitions and notations needed in the paper. The interested reader is referred
to [69,12] for a comprehensive exposition of MEL.

Membership equational logic generalizes both many-sorted and order-sorted equational logics and is essentially equivalent to Horn logic with equality. An important advantage of MEL is its great expressive power to define partial functions whose domains of definition may be characterized as sorts by axioms in the logic. In (total) MEL, partial functions are defined within a total context on kinds, but there is also an explicitly partial version of MEL not discussed here (see [69,71]). MEL has good properties as a logical framework for algebraic specification, that includes many other total and partial equational logics as special cases [69]. But this generality is achieved while keeping efficient execution under usual confluence and termination assumptions [12], as demonstrated in its Maude high-performance implementation.

Definition 2.11 A membership signature Ω is a triple (K, Σ, π) where K is a set of kinds, Σ is a K-sorted (in this context called K-kindled) algebraic signature, and π: S → K is a function that assigns to each element in its domain, called a sort, a kind. If K′ ⊆ K then Ω|K′ is the reduct membership signature obtained in the usual way.

Therefore, sorts are grouped according to kinds and operations are defined on kinds. For simplicity, we will call a “membership signature” just a “signature” whenever there is no confusion.

Definition 2.12 For any given signature Ω = (K, Σ, π) in MEL, an Ω-(membership) algebra A is a Σ-algebra together with a set As ⊆ Aπ(s) for each sort s ∈ S, and an Ω-morphism h: A → B is a Σ-morphism such that for each s ∈ S we have hπ(s)(As) ⊆ Bs. The morphism h is surjective iff it is surjective on both kinds and sorts. We let MAlgΩ denote the category of Ω-algebras and Ω-morphisms.

Definition 2.13 Given a signature Ω and a K-indexed set of variables, an atomic (Ω, X)-equation has the form t = t′, where t, t′ ∈ TΣ,K(X), and an atomic (Ω, X)-membership has the form t : s, where s is a sort and t ∈ TΣ,π(s)(X). An Ω-sentence in MEL has the form (∀X) at if at1 ∧ ... ∧ atn, where at, at1, ..., atn are atomic (Ω, X)-equations or (Ω, X)-memberships, and {at1, ..., atn} is a set (no duplications). If n = 0, then the Ω-sentence is called unconditional and written (∀X) at.

Definition 2.14 Given an Ω-algebra A and a K-kindled map θ: X → A, then A, θ ⊨Ω t = t′ iff θ(t) = θ(t′), and A, θ ⊨Ω t : s iff θ(t) ∈ As. A satisfies (∀X) at if at1 ∧ ... ∧ atn, written A ⊨Ω (∀X) at if at1 ∧ ... ∧ atn, iff for each θ: X → A, if A, θ ⊨Ω at1 and ... and A, θ ⊨Ω atn then A, θ ⊨Ω at.

Definition 2.15 An Ω-specification (or Ω-theory) T = (Ω, F) in MEL consists of a signature Ω and a set F of Ω-sentences. An Ω-algebra A satisfies (or is a model of) T = (Ω, F), written A ⊨ T, iff it satisfies each sentence in F. We let MAlgT denote the full subcategory of MAlgΩ of membership Ω-algebras satisfying an Ω-theory T.
**MEL** can be organized as a liberal institution. We let **Sign** denote the category of membership signatures, where a morphism \( \phi: (K_1, \Sigma_1, (\pi_1: S_1 \rightarrow K_1)) \rightarrow (K_2, \Sigma_2, (\pi_2: S_2 \rightarrow K_2)) \) of membership signatures is a morphism of many-kind signatures \( \phi: (K_1, \Sigma_1) \rightarrow (K_2, \Sigma_2) \) together with a map \( \phi: S_1 \rightarrow S_2 \) such that for any sort \( s \in S_1 \), \( \pi_2(\phi(s)) = \phi(\pi_1(s)) \). The sentence functor \( \text{Sen}_{MEL} \) is defined following Definition 2.13 where if \( \phi: \Omega_1 \rightarrow \Omega_2 \) is a morphism then \( \text{Sen}_{MEL}(\phi) \) is the function taking an \( \Omega_1 \)-sentence \( \gamma := (\forall X) at \) to the \( \Omega_2 \)-sentence \( \phi(\gamma) := (\forall \phi(X)) \phi(at) \) if \( \phi(at_1) \wedge \ldots \wedge \phi(at_n) \), where \( \phi(X) \) is the variable declaration \( \{x: \phi(k) | x: k \in X\} \), \( \phi(t = t') \) is the atomic \( \Omega_2, \phi(X) \)-equation \( \phi(t) = \phi(t') \) and \( \phi(t: s) \) is the atomic \( \Omega_2, \phi(X) \)-membership \( \phi(t: \phi(s)) \) with each \( x: \phi(s) \) and each operation \( \sigma \) replaced by \( \phi(\sigma) \). The model functor \( \text{Mod}_{MEL} \) associates to each membership signature \( \Omega \) its category \( \text{MA}_{\Omega} \) of membership \( \Omega \)-algebras, and if \( \phi: \Omega_1 \rightarrow \Omega_2 \) is a signature morphism, then \( \text{Mod}_{MEL}(\phi) \) is the reduct functor, which admits a left adjoint. We let \( \text{Th}_{MEL} \) denote the category of theories and morphisms of theories of \( MEL \), as defined for any institution in Subsection 2.1.

**MEL** admits complete deduction:

1. **Reflexivity**:
   \[ F \vdash \Omega \ (\forall X) t = t \]

2. **Symmetry**:
   \[ F \vdash \Omega \ (\forall X) t = t' \]
   \[ F \vdash \Omega \ (\forall X) t' = t \]

3. **Transitivity**:
   \[ F \vdash \Omega \ (\forall X) t = t', \ F \vdash \Omega \ (\forall X) t' = t'' \]
   \[ F \vdash \Omega \ (\forall X) t = t'' \]

4. **Congruence**:
   \[ F \vdash \Omega \ (\forall X) t = t' \]
   \[ F \vdash \Omega \ (\forall X, W) \sigma(W, t) = \sigma(W, t'), \text{ for each } \sigma \in \Sigma \]

5. **Membership**:
   \[ F \vdash \Omega \ (\forall X) t = t', \ F \vdash \Omega \ (\forall X) t : s \]
   \[ F \vdash \Omega \ (\forall X) t' : s \]

6. **Modus Ponens**:
   \[ \begin{cases} \text{Given a sentence in } F \\ (\forall Y) t = t' \text{ if } t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \wedge \w_1 : s_1 \wedge \ldots \w_m : s_m \text{ (resp. } (\forall Y) t : s \text{ if } t_1 = t'_1 \wedge \ldots \w_n = t'_n \w_1 : s_1 \wedge \ldots \w_m : s_m \text{ and } \theta: Y \rightarrow T_{\Sigma}(X) \text{ s.t. for all } i \in \{1, \ldots, n\} \text{ and } j \in \{1, \ldots, m\} \text{ and } F \vdash \Omega \ (\forall X) \theta(t_i) = \theta(t'_i), \ F \vdash \Omega \ (\forall X) \theta(w_j) : s_j \text{ } F \vdash \Omega \ (\forall X) \theta(t) = \theta(t') \end{cases} \]

---

3 See [69], where the rule of congruence is stated in a somewhat different but equivalent way; see also the more general rules there yielding a complete deduction system to derive not only atomic sentences, but also general \( \Omega \)-sentences (Horn clauses).
Therefore, two theories \((\Omega, F)\) and \((\Omega, F')\) have the same models if and only if \(F \vdash f'\) for each \(f' \in F'\) and \(F' \vdash f\) for each \(f \in F\).

Maude \cite{20} is an executable specification language supporting membership equational logic and also rewriting logic \cite{68}. To make specifications easier to read, and to emphasize that order-sorted specifications are a special case of membership equational ones, the following syntactic sugar conventions are widely accepted and supported by Maude:

**Subsorts.** Given sorts \(s, s'\) with \(\pi(s) = \pi(s') = k\), the declaration \(s < s'\) is syntactic sugar for the conditional membership \((\forall x : k) x : s' \text{ if } x : s\).

**Operations.** If \(\sigma \in \Omega_{k_1 \ldots k_n} k\) and \(s_1, \ldots, s_n, s \in S\) with \(\pi(s_1) = k_1, \ldots, \pi(s_n) = k_n\), then the declaration \(\sigma : s_1 \cdots s_n \rightarrow s\) is syntactic sugar for \((\forall x_1 : k_1, \ldots, x_n : k_n) \sigma(x_1, \ldots, x_n) : s \text{ if } x_1 : s_1 \land \ldots \land x_n : s_n\).

**Variables.** \((\forall x : s, X) a\) if \(a_1 \land \ldots \land a_n\) is syntactic sugar for the \(\Omega\)-sentence \((\forall x : \pi(s), X) a\) if \(a_1 \land \ldots \land a_n \land x : s\). With this, the operation declaration \(\sigma : s_1 \cdots s_n \rightarrow s\) is equivalent to \((\forall x_1 : s_1, \ldots, x_n : s_n) \sigma(x_1, \ldots, x_n) : s\).

**Example 2.16** The following Maude theory is a membership specification for the theory of small categories, where the identity morphism on an object is identified with the object itself. Note that kinds are defined implicitly as equivalence classes of sorts under the equivalence relation generated by the subsort order. Thus, \([\Arrow]\) denotes the kind to which the sort \(\Arrow\) belongs (sort \(\Object\) also belongs to \([\Arrow]\)).

```maude
fth CATEGORY is
    sorts Object Arrow .
    subsort Object < Arrow .
    ops s t : Arrow -> Object .
    op _;_ : Arrow Arrow -> [Arrow] .
    var O : Object . vars A A' A'' : Arrow .
    eq s(O) = O .
    eq t(O) = O .
    ceq O ; A = A if s(A) = O .
    ceq A ; O = A if t(A) = O .
    cmb A ; A' : Arrow if t(A) = s(A') .
    ceq t(A) = s(A') if A ; A' : Arrow .
    ceq s(A ; A') = s(A) if t(A) = s(A') .
    ceq t(A ; A') = t(A') if t(A) = s(A') .
    ceq (A ; A') ; A'' = A ; (A' ; A'') if t(A) = s(A') \ / t(A') = s(A'') .
endfth
```

Like other languages in the OBJ family \cite{53}, in Maude all the variables can be declared once in a module and then reused in each sentence. The keywords \texttt{mb} and \texttt{cmb} introduce membership and conditional membership sentences. The important thing to notice is that the composition of two arrows is indeed an arrow if and only if the target of the first arrow equals the source of the second, that is, only if the arrows are indeed composable. In fact, partiality is very naturally and elegantly supported by membership algebra (for presentations of partial membership equational logic and its relation to total MEL see \cite{69,71}).
3 Hidden Membership Algebra

An important goal of this paper is to combine hidden algebra and membership algebra. There have been some proposals to define hidden order-sorted algebra [18, 46, 66]. Since order-sorted algebra can be regarded as a special case of membership algebra, and since membership equational logic is an equational formalism that is general enough to naturally include most of the other equational formalisms in current use that we are aware of, including the partial ones [69, 71], the combination of the hidden and membership algebra frameworks seems attractive as a general way to enrich algebraic specifications with support for behavioral specification and reasoning. Besides that, since Maude already implements very efficiently membership equational logic and since it is reflective and provides support for meta-level programming, it turns out to be a convenient framework to quickly implement and prototype the hidden inference concepts presented in what follows and further discussed in Section 7.

We have not made any attempt to implement behavioral membership equational logic (BMEL) yet, but in the rest of the paper we present a language design extending Maude with behavioral modules, called BMaude, which we will illustrate with examples and will describe in greater detail in Section 5. Furthermore, in Sections 6 and 7 we present a general method to implement BMaude in Maude by reflection, and discuss the reflective implementation of behavioral deduction methods.

3.1 Basic Definitions

Definition 3.1 Given disjoint sets $K_V, K_H$ of visible and hidden kinds, a loose-data hidden membership $(K_V, K_H)$-signature $\Omega$ is a membership $(K_V \cup K_H)$-signature. A fixed-data hidden membership $(K_V, K_H)$-signature is a pair $(\Omega, D)$ where $\Omega$ is a loose-data hidden membership $(K_V, H_H)$-signature and $D$, called the data algebra, is a membership $\Omega\mid_{K_V}$-algebra. A loose-data hidden membership subsignature of $\Omega$ is a loose-data hidden membership $(K_V, K_H)$-signature $\Gamma$ with $\Gamma \subseteq \Omega$ and $\Gamma\mid_{K_V} = \Omega\mid_{K_V}$. A fixed-data hidden membership subsignature of $(\Omega, D)$ is a fixed-data hidden membership $(K_V, K_H)$-signature $(\Gamma, D)$ over the same data with $\Gamma \subseteq \Omega$ a loose-data hidden membership subsignature.

As before, we may write “hidden signature” instead of “loose-data hidden membership $(K_V, K_H)$-signature” or “fixed-data hidden membership $(K_V, K_H)$-signature,” and just write $\Omega$ instead of $(\Omega, D)$.

Definition 3.2 A loose-data hidden membership $\Omega$-algebra $A$ is a membership $\Omega$-algebra $A$, and a fixed-data hidden membership $(\Omega, D)$-algebra $A$ is a membership $\Omega$-algebra $A$ such that $A\mid_{\Omega\mid_{K_V}} = D$. A loose-data morphism of hidden algebras is any $\Omega$-morphism, while a fixed-data morphism is one which is the identity on the visible kinds. We let $\text{HMAlg}_\Omega$
and $\text{HMAlg}_{(\Omega, D)}$ denote the categories of loose-data and fixed-data hidden membership algebras, respectively.

Again, we often write just “hidden membership algebra.” Given a hidden membership subsignature $\Gamma$ of $\Omega$, the notions of $\Gamma$-context and $\Gamma$-experiment can be defined similarly to those in Definition 2.4 noticing that a context is now defined for a “kind” rather than for a “sort.” $\Gamma$-behavioral equivalence can also be defined on hidden membership $\Omega$-algebras as “indistinguishability under experiments”, but also on kinds, and appropriate notions of $\Gamma$-behaviorally congruent operation and hidden $\Gamma$-congruence can also be defined in a similar manner to those in Definition 2.5. Notice that, for a given hidden membership $\Omega$-algebra $A$ and elements $a, a' \in A_k$ for some kind $k$, it may be the case that $a \equiv_{\Omega, k} a'$ and, for some sort $s$ with $\pi(s) = k$, $a \in A_s$ while $a' \notin A_s$. In particular, elements having proper sorts can be behaviorally equivalent to elements having no proper sorts. Note that there is nothing wrong in the above: for example, $A_k$ can contain all the states of a system while $A_s$ contains all the states that can be, say, encoded in 1000 bits of memory; if the size of states’ memory is not important in defining the behavior of the system, which is usually the case, then there can be states in $A_s$ which are equivalent to states in $A_k - A_s$.

The following important result also holds, because the sorts have not been taken into consideration yet, so it can be proved just like for the many-sorted case [91, 88]:

**Theorem 3.3** Given a hidden membership subsignature $\Gamma$ of $\Omega$ and a hidden membership $\Omega$-algebra $A$, then $\Gamma$-behavioral equivalence is the largest hidden $\Gamma$-congruence on $A$.

Sentences in behavioral membership equational logic are defined exactly like those in membership equational logic. We next define behavioral satisfaction of equational and membership statements. Notice that a membership $t : s$ is behaviorally satisfied if and only if for any interpretation of $t$, there is some representative in its behavioral equivalence class having the sort $s$:

**Definition 3.4** Given a hidden subsignature $\Gamma \subseteq \Omega$, an $\Omega$-algebra $A$ and a $K$-kinded map $\theta : X \to A$, then $A, \theta \models^\Gamma \theta(t) \equiv (t')$ if and only if $\theta(t) \equiv^\Omega (t')$, and $A, \theta \models^\Gamma t : s$ if and only if $\theta(t) \equiv^\Omega \pi(a)$ for some $a \in A_s$. $A \Gamma$-behaviorally satisfies $(\forall X)$ at if $\text{at}_1 \wedge \ldots \wedge \text{at}_n$, written $A \models^\Gamma (\forall X)$ at if $A_{\text{at}_1} \wedge \ldots \wedge A_{\text{at}_n}$, iff for each $\theta : X \to A$, if $A, \theta \models^\Gamma \text{at}_1$ and ... and $A, \theta \models^\Gamma \text{at}_n$ then $A, \theta \models^\Gamma \text{at}$.

**Definition 3.5** A behavioral (or hidden) membership $\Omega$-specification (or -theory) is a triple $(\Omega, \Gamma, F)$ where $\Omega$ is a hidden membership signature, $\Gamma$ is a hidden membership subsignature of $\Sigma$, and $F$ is a set of $\Sigma$-sentences. The operations in $\Gamma - \Omega[V]$ are called behavioral. A hidden membership $\Omega$-algebra $A$ behaviorally satisfies $B = (\Omega, \Gamma, F)$ iff $A \models^\Gamma F$. We let $\text{HMAlg}_B$ and $\text{HMAlg}_{(\Omega, D)}$ denote the full subcategories of $\text{HMAlg}_\Omega$ and $\text{HMAlg}_{(\Omega, D)}$. 
respectively, of hidden membership algebras behaviorally satisfying \( B \). An operation \( \sigma \in \Omega \) is behaviorally congruent for \( B \) iff \( \sigma \) is behaviorally congruent for every \( A \models B \).

We use the same notational conventions as for hidden many-sorted algebra and the same syntactic sugar conventions as for membership algebra. As in hidden logics, if \( B = (\Omega, \Gamma, F) \) is a behavioral membership specification then all the operations in \( \Gamma \) and all the hidden constants are behaviorally congruent for \( B \).

### 3.2 The Institution of Behavioral Membership Equational Logic

Like for hidden algebra [51], one can organize hidden membership algebra in two different institutions, depending on whether the declarations of behavioral operations are regarded as part of the signature or as special sentences, and can show that there is a theoroidal forward morphism [52] between the two institutions. Also, each of the two institutions can be organized as fixed-data or loose-data. We next focus on the institution in which the declarations of behavioral operations are part of signatures and in which the data is loose; the fixed-data version can be easily derived. We call this institution \( \text{BMEL} \), for “behavioral membership equational logic.”

**Signatures**

The category \( \text{Sign}_{\text{BMEL}} \) has inclusions \( \Gamma \hookrightarrow \Omega \) of hidden membership signatures as objects; we denote these objects as pairs \((\Omega, \Gamma)\). Notice that, by Definition 3.1, \( \Omega \) and \( \Gamma \) have the same visible and hidden kinds, and that \( \Gamma \upharpoonright K^1_V = \Omega \upharpoonright K^1_V \). Given two hidden membership signatures, \((\Omega_1, \Gamma_1)\) with visible kinds \( K^1_V \) and hidden kinds \( K^1_H \) and \((\Omega_2, \Gamma_2)\) with visible kinds \( K^2_V \) and hidden kinds \( K^2_H \), respectively, a morphism of hidden membership signatures \( \phi: (\Omega_1, \Gamma_1) \rightarrow (\Omega_2, \Gamma_2) \) is a morphism of membership signatures \( \phi: \Omega_1 \rightarrow \Omega_2 \) which takes visible kinds to visible kinds and hidden kinds to hidden kinds, i.e., \( \phi(K^1_V) \subseteq K^2_V \) and \( \phi(K^1_H) \subseteq K^2_H \), takes visible operations to visible operations, i.e., \( \phi(\Omega_1 \upharpoonright K^1_V) \subseteq \Omega_2 \upharpoonright K^2_V \), and has the property that if a behavioral operation \( \delta_2 \) in \( \Gamma_2 \) has an argument kind in \( \phi(K^1_H) \) then there is some behavioral operation \( \delta_1 \) in \( \Gamma_1 \) such that \( \delta_2 = \phi(\delta_1) \). The intuition behind the later requirement is that, by translating and observing a term (state) over a signature into another signature, one cannot “observe” more than within the first signature; this can be seen as a form of “behavioral encapsulation.” Notice that the composition of hidden membership signature morphisms is well-defined. Indeed, let \( \phi': (\Omega_2, \Gamma_2) \rightarrow (\Omega_3, \Gamma_3) \) be another morphism of hidden membership signatures and let \( \delta_3 \) be an operation in \( \Gamma_3 \) having an argument sort in \( (\phi; \phi')(K^1_H) \). Then \( \delta_3 \) has an argument sort in \( \phi'(K^2_H) \), so there is an operation \( \delta_2 \) in \( \Gamma_2 \) with \( \delta_3 = \phi'(\delta_2) \). Then \( \delta_2 \) also has an argument sort in \( \phi(K^1_H) \), so there is some \( \delta_1 \) in \( \Gamma_1 \) with \( \delta_2 = \phi(\delta_1) \). Therefore, \( \delta_3 = (\phi; \phi')(\delta_1) \), i.e., \( \phi; \phi' \) is a morphism of hidden signatures. The interested reader can easily check now that \( \text{Sign}_{\text{BMEL}} \)}
is indeed a category. The morphism $\phi: (\Omega_1, \Gamma_1) \to (\Omega_2, \Gamma_2)$ in $\text{Sign}_{BMEL}$ is an inclusion if and only if $\phi: \Omega_1 \to \Omega_2$ is an inclusion in $\text{Sign}_{MEL}$; notice that, in this case, $\Gamma_2|_{(K_1^V \cup K_1^H)} \subseteq \Gamma_1$.

There is (up to isomorphism) an inclusion functor of categories $\text{Sign}_{MEL} \hookrightarrow \text{Sign}_{BMEL}$ taking a signature $\Omega$ in $MEL$ to the identity $1_\Omega: \Omega \to \Omega$ (i.e., $K_V = K$ and $K_H = \emptyset$). This inclusion functor admits a right-inverse $U: \text{Sign}_{BMEL} \to \text{Sign}_{MEL}$ which “forgets” the behavioral features; more precisely, $U(\Gamma \hookrightarrow \Omega)$ is $\Omega$ with no distinction between visible and hidden kinds. Note that $U$ is not a right adjoint of the inclusion functor.

**Sentences**

Sentences are defined like in $MEL$, ignoring the operations in $\Gamma$. More precisely, $\text{Sen}_{BMEL}: \text{Sign}_{BMEL} \to \text{Set}$ is the functor $U; \text{Sen}_{MEL}$, where $U: \text{Sign}_{BMEL} \to \text{Sign}_{MEL}$ is the forgetful functor defined above and $\text{Sen}_{MEL}$ is the sentence functor $\text{Sign}_{MEL} \to \text{Set}$ of $MEL$ (see Subsection 2.3).

**Models**

Models are also defined as in $MEL$, that is, $\text{Mod}_{BMEL}: \text{Sign}_{BMEL} \to \text{Cat}^{op}$ is the functor $U; \text{Mod}_{MEL}$ where $U: \text{Sign}_{BMEL} \to \text{Sign}_{MEL}$ is the forgetful functor defined above and $\text{Mod}_{MEL}$ is the model functor $\text{Sign}_{MEL} \to \text{Cat}^{op}$ of $MEL$. Unlike in [4,57] among other places, notice that our construction allows models in which not all the operations are behaviorally congruent.

**Satisfaction Relation**

The satisfaction relation in $BMEL$ is, as expected, the behavioral satisfaction relation in Definition 3.4, i.e., $\models_{(\Omega, \Gamma)} = \models_{\Omega}^{\Gamma}$. Before we can show the satisfaction condition, we need the following important result.

**Lemma 3.6** Given a morphism $\phi: (\Omega_1, \Gamma_1) \to (\Omega_2, \Gamma_2)$, a hidden membership $\Omega_2$-algebra $A$ and elements $a, a' \in A_{\phi(k)}$ for a kind $k$ of $\Omega_1$, then $a \equiv_{\Omega_2, \phi(k)}^{\Gamma_2} a'$ if and only if $a \equiv_{\Omega_1, k}^{\Gamma_1} a'$, where $\equiv_{\Omega_1, k}^{\Gamma_1}$ is the $\Gamma_1$-behavioral equivalence on $A|_{\Omega_1}$.

**Proof.** As shown in [88], various types of contexts and experiments generate the same behavioral equivalence. In particular, local contexts and experiments are of special interest because of their simplicity. Intuitively, a context is local if and only if it does not contain any operation whose arguments are all of visible kind. Therefore, for a hidden membership signature $(\Omega, \Gamma)$ the only local $\Gamma$-contexts for visible kinds $k_v$ are the degenerated contexts $* : k_v$ and for any local $\Gamma$-context $c$ for a hidden kind $k_h$, it is either the case that $c$ is $* : k_h$ or the case that there is some smaller (in depth) local $\Gamma$-context $c'$ and an operation $\delta \in \Gamma_{wk_h,k}$ such that $c = c'[\sigma(W, * : k_h)]$ for some appropriate variables $W$. Let $\mathcal{LE}_\Gamma[^* : k]$ denote the set of local $\Gamma$-experiments (i.e., local $\Gamma$-contexts of visible kind) for kind $k$. 18
One can relatively easily show by well-founded induction on the depth of experiments that \( \phi \) is a surjection on local experiments for kinds in \( K_2 \) that are images of kinds in \( K_1 \), that is, that for any kind \( k \in K_1 \) and any local experiment \( c_2 \in \mathcal{L}E_{\Gamma_2}[^\star : \phi(k)] \) there is some local experiment \( c_1 \in \mathcal{L}E_{\Gamma_1}[^\star : k] \) such that \( c_2 = \phi(c_1) \). Then for any two elements \( a, a' \in A_{\phi(k)} \), \( a \equiv_{\Omega_2, \phi(k)} a' \) if and only if \( A_{c_2}(a) = A_{c_2}(a') \) for any \( c_2 \in \mathcal{L}E_{\Gamma_2}[^\star : \phi(k)] \) if and only if \( A_{\phi(c_1)}(a) = A_{\phi(c_1)}(a') \) for any \( c_1 \in \mathcal{L}E_{\Gamma_1}[^\star : k] \) if and only if \( (A|_\phi)_{c_1}(a) = (A|_\phi)_{c_1}(a') \) for any \( c_1 \in \mathcal{L}E_{\Gamma_1}[^\star : k] \) if and only if \( a \equiv_{\Omega_1, k} a' \). \( \square \)

One can now show the following satisfaction condition result which completes the institution BMEL.

**Theorem 3.7** Given \( \phi : (\Omega_1, \Gamma_1) \to (\Omega_2, \Gamma_2) \) a morphism of hidden membership signatures, \( \gamma \) an \( \Omega_1 \)-sentence and \( A \) a hidden membership \( \Omega_2 \)-algebra, then \( A \models_{\Omega_2} \phi(\gamma) \) iff \( A|_{\phi} \models_{\Omega_1} \gamma \).

**Proof.** Let \( X \) be the variables declared in \( \gamma \). There is then a bijection between \([X \to (A|_{\phi})]\) and \([\phi(X) \to A]\) that associates maps \( \theta : X \to A|_{\phi} \) to maps \( \theta' : \phi(X) \to A \) related by \( \theta'(x : \phi(k)) = \theta(x : k) \). It suffices to show that for any atomic \((\Omega_1, X)\)-sentence \( \text{at} \), it is the case that \( A, \theta' \models_{\Omega_2} \phi(\text{at}) \) if and only if \( A|_{\phi}, \theta \models_{\Omega_1} \text{at} \). It is obvious that \( \theta(t) = \theta'(\phi(t)) \) for every term \( t \) in \( T_{\Omega_1}(X) \). Then for terms \( t, t' \in T_{\Omega_1}(X) \) it is the case that \( A, \theta' \models_{\Omega_2} \phi(t) = \phi(t') \) if and only if \( \theta'(\phi(t)) \equiv_{\Omega_2} \theta'(\phi(t')) \) if and only if \( \theta(t) = \theta(t') \) if and only if \( A|_{\phi}, \theta \models_{\Omega_1} t = t' \), while for a term \( t \in T_{\Omega_1,k}(X) \) and sorts \( s \) with \( \pi(s) = k \) it follows that \( A, \theta' \models_{\Omega_2} \phi(t) : \phi(s) \) if and only if there is some \( a \in A_{\phi(s)} \) such that \( a \equiv_{\Omega_2, \phi(k)} \theta'(\phi(t)) \) if and only if \( A|_{\phi}, \theta \models_{\Omega_1} t : s \). \( \square \)

It is intuitively clear that we have an extension of logics \( \text{MEL} \hookrightarrow \text{BMEL} \), in which the signatures and theories in MEL are regarded as visible in BMEL. This inclusion of logics is built along the functor \( \text{Sign}_{\text{MEL}} \hookrightarrow \text{Sign}_{\text{BMEL}} \), and it can be formalized as both a morphism and a comorphism of institutions.\(^4\) It is worth mentioning that the map \( \text{MEL} \hookrightarrow \text{BMEL} \) is conservative and essentially a sublogic in the sense of [67], because when everything is visible behavioral satisfaction becomes ordinary satisfaction. Intuitively that means that any property that holds in MEL can also be proved in BMEL, and any property of MEL that can be proved in BMEL also holds in MEL. As usual, \( \text{Th}_{\text{BMEL}} \) denotes the category of theories of BMEL. The inclusion of logics \( \text{MEL} \hookrightarrow \text{BMEL} \) induces a full subcategory inclusion \( \text{Th}_{\text{MEL}} \hookrightarrow \text{Th}_{\text{BMEL}} \), which we use later on. It is relatively easy to show that \( \text{Sign}_{\text{BMEL}} \) is cocomplete, so, by a general result in [12], the category of theories \( \text{Th}_{\text{BMEL}} \) is also cocomplete.

\(^4\) Comorphisms are also known as representations or plain maps of institutions in the literature.
3.3 Behavioral Membership Equational Deduction

The two equational deduction systems encountered so far in the paper, for hidden and for membership algebra, respectively, can be now combined in a sound equational deduction system for behavioral membership equational logic. Suppose that \( (\Omega, \Gamma, F) \) is a behavioral membership specification and let \( \models_{\Omega}^{F} \) be defined as follows:

1. Reflexivity: \( F \models_{\Omega}^{F} (\forall X) t = t \)

2. Symmetry: \( F \models_{\Omega}^{F} (\forall X) t = t' \rightarrow F \models_{\Omega}^{F} (\forall X) t' = t \)

3. Transitivity: \( F \models_{\Omega}^{F} (\forall X) t = t' \), \( F \models_{\Omega}^{F} (\forall X) t' = t'' \) \( \rightarrow F \models_{\Omega}^{F} (\forall X) t = t'' \)

4. Congruence:
\[
\begin{align*}
& a) \quad F \models_{\Omega}^{F} (\forall X, W) \sigma(W, t) = \sigma(W, t'), \text{ for each } \sigma \in \text{Der}(\Sigma) \\
& b) \quad F \models_{\Omega}^{F} (\forall X, W) \delta(W, t) = \delta(W, t'), \text{ for each congruent } \delta \in \Omega
\end{align*}
\]

5. Membership: \( F \models_{\Omega}^{F} (\forall X) t = t' \), \( F \models_{\Omega}^{F} (\forall X) t : s \) \( \rightarrow F \models_{\Omega}^{F} (\forall X) t' : s \)

6. Modus Ponens:
\[
\begin{align*}
& \text{Given a sentence in } F \\
& (\forall Y) t = t' \text{ if } t_1 = t'_1 \land \ldots \land t_n = t'_n \land w_1 : s_1 \ldots \land w_m : s_m \\
& \text{(resp. } (\forall Y) t : s \text{ if } t_1 = t'_1 \land \ldots \land t_n = t'_n \land w_1 : s_1 \ldots \land w_m : s_m) \\
& \text{and } \theta : Y \rightarrow T_{\Omega}(X) \text{ s.t. for all } i \in \{1, .., n\} \text{ and } j \in \{1, .., m\} \\
& F \models_{\Omega}^{F} (\forall X) \theta(t_i) = \theta(t'_i), \ F \models_{\Omega}^{F} (\forall X) \theta(w_j) : s_j \\
& \rightarrow F \models_{\Omega}^{F} (\forall X) \theta(t) = \theta(t')
\end{align*}
\]

Like in hidden algebra, and unlike in membership algebra, the equational inference rules above are not complete. For example, if \( B = (\Omega, \Gamma, F) \) is a behavioral specification of a memory cell where \( \Omega \) contains two sorts \( \text{Nat} \) and \( \text{Cell} \) such that \( \pi(\text{Nat}) \) is a visible kind and \( \pi(\text{Cell}) \) is a hidden kind, two operations \( \text{get} : \text{Cell} \rightarrow \text{Nat} \) and \( \text{put} : \text{Cell} \text{Nat} \rightarrow \text{Cell} \), \( \Gamma \) contains only \( \text{get} \), and \( F \) contains the obvious equation \( (\forall n : \text{Nat}, c : \text{Cell}) \text{get}(\text{put}(n, c)) = n \), then one can show that \( F \models_{\Omega}^{F} (\forall n \cdot n' : \text{Nat}, c : \text{Cell}) \text{get}(\text{put}(n, \text{put}(n', c))) = \text{get}(\text{put}(n, c)) \) but one cannot show \( F \models_{\Omega}^{F} (\forall n, n' : \text{Nat}, c : \text{Cell}) \text{put}(n, \text{put}(n', c)) = \text{put}(n, c) \). Notice that the latter equation is behaviorally satisfied by \( B \), because there
is only one experiment that can be performed, namely \( \text{get}(* : \pi(\text{Cell})) \). It is shown in [15] that the behavioral satisfaction problem is \( \Pi^0_2 \)-hard, so it is neither recursively enumerable nor co-recursively enumerable; in particular, this means that there is no complete deduction system for behavioral equivalence. Intuitively, this is because Turing machines can be encoded as finite behavioral specifications and Turing machine computations can be reduced to behavioral reasoning using the rules above; then one can encode a universal Turing machine \( U \) taking as input another Turing machine \( M \) and a string \( x \) and saying if \( M \) terminates when executed on input \( x \), and this problem is known to be \( \Pi^0_2 \)-complete.

### 3.4 Theorems of Hidden Constants and of Deduction

So far we have seen how unconditional sentences can be inferred. In order to infer a conditional equation, we need to first eliminate the quantifier variables of that equation, via the *theorem of constants*, and then to add its condition to the specification, via the *deduction theorem*. Many logicians also call the theorem of constants the "generalization theorem," and the deduction theorem the "implication elimination theorem."

**Theorem 3.8 Hidden Constants.** If \( B = (\Omega, \Gamma, F) \) is a behavioral specification, \( \gamma \) is an \( \Omega \)-sentence \((\forall Y, X) \ at \ if \ at_1 \land \ldots \land at_n \ where \ X \ contains \ only \ hidden \ variables, \ and \ \gamma_X \ is \ the \ (\Omega \cup X)\)-sentence \((\forall Y) \ at \ if \ at_1 \land \ldots \land at_n \), then \( B \ |= \gamma \iff B_X \ |= \gamma_X \), where \( B_X = (\Omega \cup X, \Gamma, F) \).

The theorem of hidden constants holds for both fixed-data and loose-data hidden membership approaches. However, notice that visible constants can be eliminated only in a loose-data setting, because otherwise they would change the data.

**Proposition 3.9 Deduction.** Given behavioral specification \( B = (\Omega, \Gamma, F) \), ground \((\Omega, \emptyset)\)-atomic sentences \( at_1, \ldots, at_n \), and an \((\Omega, X)\)-atomic sentence \( at \), let \( F' = F \cup \{ (\forall \emptyset) \ at_1, \ldots, (\forall \emptyset) \ at_n \} \), and let \( B' \) be the behavioral specification \((\Omega, \Gamma, E') \). Then \( B' \ |= (\forall X) \ at \ iff \ B \ |= (\forall X) \ at \ if \ at_1 \land \ldots \land at_n \).

### 3.5 Coinductive Reasoning

This section briefly introduces another behavioral proof technique, called *coinduction*, which, combined with behavioral membership equational reasoning presented in Subsection 3.3, becomes very powerful. The BOBJ system already supports various versions of coinduction, which gradually evolved as we found new situations where the previous versions didn’t work. We are going to use our experience gathered while developing BOBJ in the design of BMaude. In this section, we only list the various forms of coinduction that we considered so far, without proofs and without examples, and in Subsection 7.2 we further describe how circular coinductive rewriting can be implemented using
Maude’s reflective capabilities. The reader is encouraged to use coinduction to show behavioral properties of the specifications in Subsection 3.6.

3.5.1 General Coinduction
Since $\Gamma$-behavioral equivalence is the largest hidden $\Gamma$-congruence in any membership $\Omega$-algebra (see Theorem 3.3), the following method to prove that an $\Omega$-equation $(\forall X) t = t'$ is behaviorally satisfied by a specification $\mathcal{B}$ is sound:

**Coinduction Method for Equations:**

*Input:* $\mathcal{B} = (\Omega, \Gamma, F)$ and a pair of $\Omega$-terms $(t, t')$

*Step 1:* Pick an “appropriate” binary relation $R$ on terms

*Step 2:* Prove that $R$ is a hidden $\Gamma$-congruence

*Step 3:* Show that $t R t'$.

The relation $R$ is called the candidate relation and even though it may depend on the particular equation $(\forall X) t = t'$, in many situations it is the same for a large class of equations. Similarly, we can use coinduction to prove memberships $(\forall X) t : s$:

**Coinduction Method for Memberships:**

*Input:* $\mathcal{B} = (\Omega, \Gamma, F)$ and a pair $(t, s)$

*Step 1:* Pick an “appropriate” term $t'$

*Step 2:* Prove the membership $t' : s$.

*Step 3:* Pick an “appropriate” binary relation $R$ on terms

*Step 4:* Prove that $R$ is a hidden $\Gamma$-congruence

*Step 5:* Show that $t R t'$.

It is the case that sometimes, in order to define the candidate relation, the specification needs to be first extended. To be rigorous, one has to prove that the extension is conservative, that is, that any model of the initial specification can be extended to a model over the larger specification, thus providing the environment to define the candidate relation. We hope to address all these issues soon elsewhere. We refer the interested reader to [88] for many proofs by coinduction in the context of hidden algebra.

3.5.2 Cobasis Coinduction
Behavioral deduction cannot be completely automatized, because the behavioral satisfaction is $\Pi_2^0$-hard [15]. Although general coinduction is very powerful in practice, it is unfortunately hard to automate. One of our major goals in the design of BMaude is to find less powerful methods that can be automated. But first,
Definition 3.10 If $B' = (\Omega', \Gamma', F')$ is a conservative extension\(^5\) of $B = (\Omega, \Gamma, F)$ and if $\Delta \subseteq \Omega'$, then $\Delta$ is a **cobasis for $B$** iff for all hidden-kindred terms $t, t' \in T_{\Omega, k_h}(X)$, if $B' \models (\forall W, X) \delta(W, t) = \delta(W, t')$ for all appropriate $\delta \in \Delta$ then $B \models (\forall X) t = t'$.

This is a rather semantic notion (see also [51]), but there are stronger versions that seem to suffice in practice, such as **strong cobasis** (called just cobasis in [91]), and the special case of **complete set of observers** [8].

Hereafter, suppose that $\Delta$ is a cobasis of $B$, $B' = \text{Der}(\Omega)$, and $\Delta \subseteq \text{Der}(\Gamma)$, where $\text{Der}(\Omega)$ is the signature containing the $\Omega$-derived operations. We can then add one more inference rule to the six ones presented in Subsection 3.3:

(7) **$\Delta$-Coinduction:**

\[
\frac{\Gamma \vdash F (\forall W, X) \delta(W, t) = \delta(W, t') \text{ for all appropriate } \delta \in \Delta}{\Gamma \vdash F (\forall X) t = t'}
\]

We refer the interested reader to [91] for the soundness of the rule above, and to [SS] for more details and examples. [SS] also presents a criterion for detecting automatically cobases, which has been implemented in BOBJ and seems to perform quite well in practice.

3.5.3 **Circular Coinduction**

We next briefly describe circular coinduction, so called because it handles infinite recursions that escape previous rules; we may also call it circular $\Delta$-coinduction or $\Delta \triangleleft$-coinduction. Suppose that $\Delta$ is a cobasis\(^6\) as above and that $< \bigwedge \Gamma$ is a well-founded partial order on $\Gamma$-contexts preserved by $\Gamma$, such as the depth of contexts. The soundness of the following new inference rule was shown in [SS] for the hidden algebraic case, and translates easily to our hidden membership algebraic framework:

For each appropriate $\delta \in \Delta$, either there is a $\Gamma$-term $u$ s.t.

\[
\Gamma \vdash F \quad (\forall W, X) \delta(W, t) = u \quad \text{and} \quad F \vdash (\forall W, X) \delta(W, t') = u
\]

or there is some context $c < \delta$ such that

\[
\Gamma \vdash F \quad (\forall W, X) \delta(W, t) = c[t] \quad \text{and} \quad F \vdash (\forall W, X) \delta(W, t') = c[t']
\]

(8) **$\Delta \triangleleft$-Coind.:**

\[
\frac{\Gamma \vdash F (\forall W, X) \delta(W, t) = \delta(W, t') \text{ for all appropriate } \delta \in \Delta}{\Gamma \vdash F (\forall X) t = t'}
\]

BOBJ implements circular coinductive rewriting [45], an algorithm that combines the coinduction inference rules presented above with behavioral

\(^5\) I.e., $\Omega \subseteq \Omega'$ and for any hidden $\Omega$-algebra $A \models B$ there is some hidden $\Omega'$-algebra $A' \models B'$ such that $A'|_{\Omega} = A$.

\(^6\) We have only shown the soundness of circular coinduction for some special cobases called “complete sets of observers;” however, we claim that the result holds for general cobases.
rewriting, an adaptation of term rewriting to our behavioral equational deduction system. We discuss in more detail both behavioral rewriting and circular coinductive rewriting in Section 7. Using BOBJ, we could automatically prove all the reasonable statements that we knew, including all those previously done in CoClam [25] using complex heuristics. Of course, new more powerful inference rules may be needed in the future.

3.6 Examples

The following three simple, yet nontrivial, examples illustrate the expressiveness of hidden membership algebra. Some of them could not be easily expressed in hidden order-sorted algebra.

**Example 3.11** Paths on multigraphs, that is, graphs allowing multiple edges between any two nodes, can be specified as a behavioral membership theory in which the nodes, the edges and the paths are all hidden sorts, with edges being a subsort of paths, and nodes belonging to a different (hidden) kind. Following Maude’s conventions, we also assume that kinds are defined implicitly by the subsort relation in BMaude. Following conventions similar to those in BOBJ, a behavioral (functional) theory is introduced with the keywords \texttt{bfth} ... \texttt{endbfth} and all the sorts explicitly declared in a behavioral theory are assumed hidden; if one wants to declare a visible sort then one should do it in a different non-behavioral module or theory (of type \texttt{fmod} or \texttt{fth}) and then protect it. Similarly, all the operations are assumed behavioral by default. If one wants an operation to be nonbehavioral, then one should declare it using the attribute \texttt{ncong}, stating that that operation is not intended to be behaviorally congruent; semantically that means that models in which the function interpreting that operation does not preserve the behavioral equivalence generated by the behavioral operations are allowed.

```
bfth MULTI-GRAph is
  pr MACHINE-INT .
  pr STRING .
  sorts Node Edge Path .
  subsort Edge < Path .
  ops source target : Path -> Node .
  op _;_ : Edge Path -> [Path] .
  op label : Node -> String .
  op length : Path -> MachineInt .
  op head : Path -> Edge .
  op tail : Path -> [Path] .
  op * : [Path] .
  var E : Edge . var P : Path .
  cmb E ; P : Path if target(E) = source(P) .
  ceq source(E ; P) = source(E) if E ; P : Path .
  ceq target(E ; P) = target(P) if E ; P : Path .
  ceq length(E ; P) = 1 + length(P) if E ; P : Path .
  eq length(E) = 1 .
  ceq head(E ; P) = E if E ; P : Path .
```

\begin{verbatim}
  ceq tail(E ; P) = P if E ; P : Path .
  eq head(E) = E .
  eq tail(E) = * .
endbfth

The modules \texttt{MACHINE-INT} and \texttt{STRING} are builtin in Maude. Notice that
syntactic sugar conventions are used, that there are two hidden kinds, and
that all the operations are behavioral. The attribute \texttt{label} gives the label of
a node as a string, \texttt{source}, \texttt{target} and \texttt{length} give the source node, the target
node and the length of any well-formed path, and \texttt{head} and \texttt{tail} give the first
edge and the rest of a path, respectively. Then in any hidden membership
algebra of this specification, two nodes are behaviorally equivalent if and only
if they have the same label. An edge and a path can be composed, provided
that the target node of the edge is behaviorally equivalent to the source node
of the path, and if this is the case, then the source, the target, the length,
the head and the tail of the newly formed path verify the expected properties.
One can show that two paths formed by compositions of edges are behaviorally
equivalent if and only if they visit the same nodes in the same order. Notice
that there can be “unreachable” paths, i.e., paths which are not built by
compositions of edges, but we cannot assume and prove anything about those.
More sentences are needed in order to exclude them. If we had declared \texttt{head}
and \texttt{tail} non-behaviorally congruent, using the attribute \texttt{ncong}, then two
paths would have been behaviorally equivalent if and only if they start (and
also end) in behaviorally equivalent nodes, i.e., nodes having the same label,
and the same length. If, in addition, \texttt{length} were to be declared \texttt{ncong}, then
two paths would be behaviorally equivalent if and only if they start (and also
end) in equivalent nodes.

Example 3.12 A nondeterministic stack is a stack in which unknown num-
bers are pushed. One can regard a random number generator in a multiclient
environment as a nondeterministic stack, where the operating system pushes
random numbers in a stack and the various clients consume them. The clients
can therefore “observe” the stack only by using the attribute \texttt{top} and the
method \texttt{pop}; they do not have access to \texttt{push}. That means that two states
of such a stack appear to be equivalent if and only if they contain the same
elements in the same order. In an early design stage of such a random number
generator, one is not interested in how the numbers are actually generated.
That is, one does not want to specify the low-level details of a possible imple-
mentation. However, one would want to specify the crucial property that any
sound implementation of a random number generator should satisfy, namely
that in apparently equivalent situations it may (and actually should) generate
different numbers. All these requirements suggest the following behavioral
membership theory:
\begin{verbatim}
bftth NONDETERMINISTIC-STACK is
  pr MACHINE-INT .
  sort NdStack .
\end{verbatim}
\end{verbatim}

25
op top : NdStack -> [MachineInt] .
op pop : NdStack -> [NdStack] .
op push : NdStack -> NdStack [ncong] .
var S : Stack .
mb top(push(S)) : MachineInt .
eq pop(push(S)) = S .
endbfth

We have not defined any initial state on purpose, to allow a maximum of
flexibility for implementations, and have not required \texttt{top} and \texttt{pop} to be always
properly defined on any stack \footnote{This partiality is indicated by the coarity of these operations being the \textit{kinds} [MachineInt] and [NdStack], instead of the \textit{sorts} MachineInt and NdStack.} (that is because some stacks can be improperly
formed). The crucial issue to be noticed in the above behavioral theory is that
\texttt{push} is \textit{not} congruent, reflecting the intuition that different numbers can be
generated in apparently equivalent situations; in particular, models in which
\texttt{push(pop(push(S)))} is not behaviorally equivalent to \texttt{push(S)} are allowed (in
fact, these are the desired models).

\textbf{Example 3.13} Infinite streams are common in the formal specification and
verification of protocols, where they serve as inputs and outputs. Many ex-
amples with infinite streams occur in the literature; here we focus on some
simple properties of streams which seem to be hard, if not impossible, to han-
dle using hidden (many-sorted) algebra, even in its order-sorted form, but can
be naturally expressed in hidden membership algebra:

\begin{verbatim}
bfth STREAM is
  pr MACHINE-INT .
sorts Stream Constant Blink .
subsort Constant Blink < Stream .
op head : Stream -> MachineInt .
op tail : Stream -> Stream .
op zip : Stream Stream -> Stream .
ops zero one : -> Constant .
ops zero-one one-zero : -> Blink .
vars S S' : Stream .
vars C C' : Constant .
var B : Blink .
eq head(zip(S,S')) = head(S) .
eq tail(zip(S,S')) = zip(S',tail(S)) .
eq tail(C) = C .
cmb S : Constant if tail(S) = S .
eq tail(tail(B)) = B .
cmb S : Blink if tail(tail(S)) = S .
eq head(zero) = 0 .
eq head(one) = 1 .
eq head(zero-one) = 0 .
eq tail(zero-one) = one-zero .
eq head(one-zero) = 1 .
eq tail(one-zero) = zero-one .
endbfth
\end{verbatim}
As usual, `head` and `tail` give the head and the tail of a stream, and `zip` merges two streams. All the operations are behavioral, but as in [91,88] one can (automatically) show that `zip` is behaviorally congruent with respect to the behavioral equivalence generated by just `head` and `tail`, so \{`head`, `tail`\} form a cobasis for `STREAM`. A stream is constant if and only if it is behaviorally equivalent to its tail and is blink if and only if it is equivalent to the tail of its tail. `zero` and `one` are constant streams of 0s and 1s, respectively, and `zero-one` and `one-zero` are blink streams repeating 01 and 10, respectively. One should be then able to prove the following properties in BMaude:

\[
\begin{align*}
\text{eq zero-one} &= \text{zip}(\text{zero}, \text{one}) . \\
\text{eq one-zero} &= \text{zip}(\text{one-zero}) . \\
\text{subsort Constant} &< \text{Blink} . \\
\text{mb zip}(C, C') : \text{Blink} .
\end{align*}
\]

The first two behavioral equalities can be proved by circular coinductive rewriting, like in BOBJ, while the last two conditional memberships could be proved using the theorems of hidden constants and of deduction for hidden membership algebra (see Subsection 3.4).

### 3.7 Other Notions of Behavioral Equivalence

The notion of behavioral equivalence in Subsection 3.1 may be considered by some readers a bit too restrictive, in the sense that two elements are equivalent only if they are equal under any experiment, including those that do not return a properly sorted (but just a kinded) value. There are situations in which it makes sense to consider behavioral equivalence only “up to sortedness”, that is, to ignore the results of experiments that do not have a proper (visible) sort. It turns out that three other notions of behavioral equivalence make actually perfect sense in the context of membership algebra, which, due to their partial algebraic flavour, we call existential, strong, and weak. In this subsection we discuss these notions, give intuitive examples for each, and claim some properties for them that we find worth investigating in more detail in the near future.

#### 3.7.1 Hidden Partial Membership Algebra

It is best to give the intuitions for these notions of behavioral equivalence in the context of partial membership algebra [69] first. Without giving all the formal definitions and results, which are fairly easy to adapt, we then tacitly consider a hidden version of partial membership algebra, using the same general notations for signatures and sorts as for the total case, but in which some of the operations in \(\Omega\), including the behavioral ones in \(\Gamma\), can be partial. What is the right notion of behavioral equivalence in such a partial setting?

**Definition 3.14** Given a hidden partial membership \(\Omega\)-algebra \(A\) and two elements \(a, a' \in A_k\), then \(a, a'\) are
• existentially $\Gamma$-behaviorally equivalent, written $a \equiv^\exists_{\Omega, e} a'$ iff for any $\Gamma$-experiment $c$ and any $(K_V \cup K_H)$-kinded map $\theta: \text{var}(c) \to A$, both $A_c(a)(\theta)$ and $A_c(a')(\theta)$ are defined and are equal;

• strongly $\Gamma$-behaviorally equivalent, written $a \equiv^\exists_{\Omega, s} a'$ iff for any $\Gamma$-experiment $c$ and any $(K_V \cup K_H)$-kinded map $\theta: \text{var}(c) \to A$, both $A_c(a)(\theta)$ defined iff $A_c(a')(\theta)$ defined, and if this the case then they are equal;

• weakly $\Gamma$-behaviorally equivalent, written $a \equiv^\exists_{\Omega, w} a'$ iff for any $\Gamma$-experiment $c$ and any $(K_V \cup K_H)$-kinded map $\theta: \text{var}(c) \to A$, if both $A_c(a)(\theta)$ and $A_c(a')(\theta)$ are defined then they are equal.

To illustrate these notions, consider a simple hidden partial equational theory of infinite streams of booleans $\{0, 1\}$, with one visible sort $\text{Bool}$ and one hidden sort $\text{Stream}$, in which $\Omega = \Gamma$ consist of the partial operations $\text{head}: \text{Stream} \to \text{Bool}$ and $\text{tail}: \text{Stream} \to \text{Stream}$. For any hidden partial membership algebra model $A$ of this specification and any element $a \in A_{\text{Stream}}$, we can associate to $a$ an infinite list of elements in $\{0, 1, \perp\}$ ($\perp$ is a special symbol used for “undefined”), say $\text{beh}(a)$, where $\text{beh}(a)_i$ is the result of the experiment giving the $i$-th element of $a$, that is, $A_{\text{head}(\text{tail}^i(a))}(a)$. Then two streams $a,a' \in A_{\text{Stream}}$ are existentially behaviorally equivalent if and only if $\text{beh}(a)$ and $\text{beh}(a')$ are identical and contain only symbols 0 and 1; they are strongly behaviorally equivalent if and only if $\text{beh}(a)$ and $\text{beh}(a')$ are identical, and they are weakly behaviorally equivalent if and only if $\text{beh}(a)$ and $\text{beh}(a')$ do not disagree on positions which are defined in both. More intuitively, one can think of $a$ as a specification of a system/function and of $a'$ as an implementation of that system/function, and of $\text{beh}(a)$ and $\text{beh}(a')$ as the requirements and the actual behavior of that system/function, respectively. As expected, $a \perp$ in $\text{beh}(a)$ is an unspecified behavior while $a \perp$ in $\text{beh}(a')$ is an unimplemented behavior; then $a$ and $a'$ are existentially equivalent if and only if the specification is total and is complete and correct for the implementation, are strongly equivalent if and only if the specification is complete and correct for the implementation (but both can have undefined behaviors), and are weakly equivalent if and only if the implementation and the specification agree on their commonly defined behaviors, i.e., they do not disagree. We think that all these three notions of behavioral equivalence relations are natural and make sense independently of each other.

Based on these three notions of equivalence, one can define appropriate behavioral satisfaction relations, similar to that in Definition 3.4.

3.7.2 Behavioral Equivalences in Hidden Membership Algebra

With the intuitions developed in the previous subsection, we can now define similar notions of behavioral equivalence for hidden (total) membership algebra.

Definition 3.15 Given a hidden membership $\Omega$-algebra $A$ and two elements $a,a' \in A_k$, then $a,a'$ are
- **existentially \( \Gamma \)-behaviorally equivalent**, written \( a \equiv_{\Omega,e}^\Gamma a' \), iff for any \( \Gamma \)-experiment \( c \) and any \((K_V \cup K_H)\)-kinded map \( \theta : var(c) \to A \), there is some visible sort \( v \) such that \( A_c(a)(\theta) = A_c(a')(\theta) \in A_v \);
- **strongly \( \Gamma \)-behaviorally equivalent**, written \( a \equiv_{\Omega,s}^\Gamma a' \), iff for any \( \Gamma \)-experiment \( c \), any \((K_V \cup K_H)\)-kinded map \( \theta : var(c) \to A \), and any visible sort \( v \), if \( A_c(a)(\theta) \in A_v \) or \( A_c(a')(\theta) \in A_v \) then \( A_c(a)(\theta) = A_c(a')(\theta) \);
- **weakly \( \Gamma \)-behaviorally equivalent**, written \( a \equiv_{\Omega,w}^\Gamma a' \), iff for any \( \Gamma \)-experiment \( c \) and any \((K_V \cup K_H)\)-kinded map \( \theta : var(c) \to A \), if \( A_c(a)(\theta) \in A_v \) and \( A_c(a')(\theta) \in A_{v'} \) for visible sorts \( v \) and \( v' \) then \( A_c(a)(\theta) = A_c(a')(\theta) \).

One can now define three appropriate notions of behavioral satisfaction, namely existential, strong and weak, denoted by \( \models_{\Omega,e}^\Gamma \), \( \models_{\Omega,s}^\Gamma \), and \( \models_{\Omega,w}^\Gamma \), respectively. Notice that if a behavioral specification is given in such a way that, with the syntactic sugar conventions, the sorts of any visible kind form a filtered set, i.e., any pair of sorts is majored by another sort, and if \( A \) satisfies the sentences associated to these syntactic sugar conventions, then \( a \equiv_{\Omega,w}^\Gamma a' \) iff for any \( \Gamma \)-experiment \( c \), any \((K_V \cup K_H)\)-kinded map \( \theta : var(c) \to A \), and any visible sort \( v \), if \( A_c(a)(\theta), A_c(a')(\theta) \in A_v \) then \( A_c(a)(\theta) = A_c(a')(\theta) \).

The behavioral equivalence notions above, and their relationships to the similar notions defined for hidden partial membership algebra, are more subtle than it might seem. This is because, unlike in partial algebra, in membership algebra a term can be evaluated to an element having a sort even if not all its subterms have the same property. More precisely, a value \( A_c(a)(\theta) \) as above can belong to \( A_v \) for some visible sort \( v \) even if not all of \( A_{c'}(a)(\theta) \), for \( c' \) a subcontext of \( c \), belong to a sort carrier of \( A \). It might be the case that, as we advance this research and explore deeper relationships with hidden partial membership algebra, the above definitions will change by requiring all the subterms of the experiments to also be defined, as we did in \[69,71\], using some appropriate notion of (perhaps “behavioral”) envelope. We also think that, under appropriate requirements of envelope invariance similar to those in \[71\], there should be an “almost zero representational distance” between theories in BPME and BMEL, which should allow one to safely regard the same theory as either partial or total depending on the purposes at hand.

We are still experimenting with the four notions of behavioral equivalence and satisfaction for hidden membership algebra presented in this paper. It is not clear to us yet which is the best to use in practice and which has the most elegant properties, but in the rest of the paper we adopt the one which was considered first, which we just call “behavioral equivalence” from now on.

### 4 Coalgebraic Hidden Membership Algebra

In this section we investigate a theoretically important special case of fixed-data hidden membership algebra, which has the property that, given a hidden membership signature (or theory), the category of its hidden membership
algebras admits a final model and is isomorphic to a category of coalgebras.

4.1 Final Models and Coalgebras for Signatures

We first define a class of fixed-data hidden membership signatures which are coalgebraic, in the exact sense that their models are isomorphic to a category of coalgebras admitting a final model.

**Definition 4.1** A coalgebraic hidden membership signature is a fixed-data hidden membership \((K_V, K_H)\)-signature with the property that each operation has at most one argument of hidden kind. The other notions remain unchanged, except that \(\Gamma\) is assumed equal to \(\Omega\) in any behavioral theory \((\Omega, \Gamma, F)\); for this reason, we omit \(\Gamma\) in the rest of this section. We call this special case of behavioral membership equational logic coalgebraic behavioral membership algebra.

In the rest of this section we assume a coalgebraic hidden membership \((K_V, K_H)\)-signature \((\Omega, D)\), where \(\Omega = (K_V \cup K_H, \Sigma, \pi)\).

**Definition 4.2** We let \(\text{Set}^H_\pi\) denote the category whose objects are \(K_H\)-indexed sets \(A = \{A_{k_h} \mid k_h \in K_H\}\) together with a subset \(A_{s_h} \subseteq A_{k_h}\) for each \(s_h \in S\) with \(\pi(s_h) = k_h\), and whose morphisms are \(K_H\)-indexed functions \(f: A \to B\) such that \(f_{\pi(s)}(A_s) \subseteq B_s\) for each \(s \in S\).

Intuitively, the objects in \(\text{Set}^H_\pi\) will be acting as the carriers of hidden kind of fixed-data hidden membership \((\Omega, D)\)-algebras.

**Definition 4.3** Let \(\mathcal{G}_{(\Omega, D)}: \text{Set}^H_\pi \to \text{Set}^H_\pi\) denote the functor which is defined on objects as follows:

- \(\mathcal{G}(A)_{k_h} = (\prod_{\sigma: wk_h \to k_v} [D^w \to D_{k_v}]) \times (\prod_{\sigma: wk_h \to k_{h'}} [D^w \to A_{k_{h'}}])\) for each \(k_h \in K_H\),
- \(\mathcal{G}(A)_{s_h} = \mathcal{G}(A)_{k_h}\) for each \(k_h \in K_H\) and \(s_h \in S\) with \(\pi(s_h) = k_h\),

and on morphisms as

- \(\mathcal{G}(g)_{k_h}((\{f_\sigma\}_{\sigma: wk_h \to k_v}, \{f_\sigma\}_{\sigma: wk_h \to k_{h'}})) = (\{f_\sigma\}_{\sigma: wk_h \to k_v}, \{f_\sigma g_{k_{h'}}\}_{\sigma: wk_h \to k_{h'}})\) for each indexed function \(g: A \to B\) in \(\text{Set}^H_\pi\).

One can easily show that \(\mathcal{G}_{(\Omega, D)}: \text{Set}^H_\pi \to \text{Set}^H_\pi\) is indeed a functor. The following is an important property that gives hidden membership algebra its coalgebraic flavor. Its proof follows the same pattern as the one for hidden many-sorted algebra [48], so we do not repeat it here:

**Theorem 4.4** \(\text{HMAlg}_{(\Omega, D)}\) is isomorphic to \(\text{CoAlg}(\mathcal{G}_{(\Omega, D)})\).

The functor \(\mathcal{G}_{(\Omega, D)}\) has all the nice properties of polynomial functors [94], so the following important corollary holds:
Corollary 4.5 HMAlg(Ω,D) has a final object, denoted \(Z_{(Ω,D)}\). Given any hidden membership \((Ω,D)\)-algebra \(A\), the behavioral equivalence on \(A\) is exactly the kernel of the unique morphism \(A \to Z_{(Ω,D)}\).

Proof. (Hint) The final model either can be constructed directly, as in [48], making use of local contexts, or can be shown to exist like in [94]; in both situations, \(Z_{(Ω,D),s} = Z_{(Ω,D),π(s)}\) for each \(s \in S\) with \(π(s) ∈ K_H\).

Notice that the fact that operations have at most one argument of hidden kind plays a crucial role in the results above. Like in [15,88], one can show using a set theory cardinality argument that the category of hidden membership algebras over a hidden membership signature containing a binary method (an operation with two arguments of hidden kind) does not admit final models.

4.2 Coalgebraic Theories

Sentences can influence the coalgebraic nature of a hidden membership signature significantly. Unlike in standard membership equational logic, a behavioral theory having the natural numbers as data, and containing one attribute \(a\) and the equation \((∀x : h) a(x) = a(x) + 1\) admits only one degenerated model, which has the hidden carrier empty, while a similar theory containing the equation \((∀∅) 0 = 1\) admits no models. The consistency problem for a behavioral theory seems to be undecidable in general (or more precisely, \(Π_2^0\)-hard) even in its simplified hidden (co)algebraic form [15,88]. We are not going to develop either the general consistency problem nor the coalgebraic problem\(^8\) here because they are not needed for the design of BMaude, but we hope to do so soon elsewhere.

In this subsection we instead focus on sufficient conditions for a fixed-data behavioral membership theory to admit final models. More precisely, we show that final models exist whenever the theory is consistent and its sentences, both equations and memberships, contain at most one hidden variable. We make use of the following result proved in [85,89,88]:

**Lemma 4.6** Let \(\mathcal{C}\) be a category admitting coproducts, a final object \(Z_{\mathcal{C}}\), and a well-powered inclusion system\(^9\), and let \(\mathcal{K}\) be a class of objects in \(\mathcal{C}\). If \(\mathcal{K}\) is closed under coproducts and quotients then it has a final object \(Z_{\mathcal{K}}\) which is a subobject of \(Z_{\mathcal{C}}\).

We next take \(\mathcal{C}\) to be the category HMAlg(Ω,D) where \((Ω,D)\) is a coalgebraic hidden membership signature, which has all the properties required by the above lemma.

---

\(^8\) That is: given a behavioral membership theory as input, is it the case that its category of models is isomorphic to a category of coalgebras?

\(^9\) An inclusion system is a concept similar to factorization systems. The result also holds for factorization systems.
Definition 4.7 If $F$ is a set of $\Omega$-sentences then $\text{HMAAlg}_{(\Omega,D,F)}$ denotes the full subcategory of $\text{HMAAlg}_{(\Omega,D)}$ of hidden membership $(\Omega,D)$-algebras that behaviorally satisfy $F$. Notice that $\text{HMAAlg}_{(\Omega,D,F)}$ can be empty.

Theorem 4.8 Let $(\Omega,D)$ be a coalgebraic hidden membership signature and let $F$ be a set of $\Omega$-sentences such that each sentence in $F$ contains at most one hidden variable. Then $\text{HMAAlg}_{(\Omega,D,F)}$ admits a final model whenever it is nonempty.

Proof. First, notice that the consistency requirement is needed because one can, for example, add equations which are not satisfied by $D$, in which case $B$ does not have models at all. However, the problem of checking whether the algebra $D$ satisfies a given visible equation can be arbitrarily complex, so one cannot hope to have a general, complete, data consistency checker. For simplicity, in the rest of the paper we will assume that all our behavioral specifications are consistent.

We use Lemma 4.6. Let $C$ be the category $\text{HMAAlg}_{(\Omega,D)}$ and let $K$ be $\text{HMAAlg}_{(\Omega,D,F)}$. We have to show that $K$ is closed under coproducts and surjections. Let $\gamma = (\forall X)\; \text{at} \; at_1 \land \ldots \land at_n$ be an equation or a membership sentence in $F$. Let us first consider a family of models $\{A_i\}_{i \in I}$ in $\text{HMAAlg}_{(\Omega,D,F)}$ and let $\theta: X \to A$ be any map, where $A := \bigsqcup_{i \in I} A_i$ is the hidden membership algebra whose hidden carriers are the disjoint unions of the individual hidden carriers of the algebras $A_i$. Notice that the behavioral equivalence on $A$ is also the disjoint union of the individual behavioral equivalences on the algebras $A_i$. Since $X$ contains at most one hidden variable and since all the algebras $A_i$ have the same data $D$, there is an $i \in I$ and a map $\theta_i: X \to A_i$ such that $\theta$ factors through $\theta_i$. The satisfaction $A, \theta \models^\Omega_\gamma$ follows then from $A_i, \theta_i \models^\Omega_\gamma$, $i \in I$, so $K$ is closed under coproducts. Notice next that, by Corollary 4.5, stating that the behavioral equivalence on an $(\Omega,D)$-algebra is exactly the kernel of the unique morphism to the final $(\Omega,D)$-algebra, if $g: A \to B$ is a morphism of $(\Omega,D)$-algebras, then two elements $a,a'$ are behaviorally $\Omega$-equivalent in $A$ if and only if $g(a), g(a')$ are behaviorally $\Omega$-equivalent in $B$. Let us now consider a surjective morphism of hidden membership $(\Omega,D)$-algebras $e: A \to B$ such that $A$ satisfies $F$, and let $\theta_B: X \to B$ be any map. By the surjectivity of $e$, there is some map $\theta_A: X \to A$ such that $\theta_B = \theta_A; e$. In order to show that $B$ satisfies $F$, it suffices to show that for any atomic $(\Omega,X)$-sentence $at'$ it is the case that $A, \theta_A \models^\Omega_\Omega at'$ if and only if $B, \theta_B \models^\Omega_\Omega at'$. If $at'$ is an atomic $(\Omega,X)$-equation $t = t'$ then it follows by the note above that $\theta_A(t) \equiv_A \theta_A(t')$ if and only if $e(\theta_A(t)) \equiv_B e(\theta_A(t'))$, i.e., if and only if $\theta_B(t) \equiv_B \theta_B(t')$. If $at'$ is an atomic $(\Omega,X)$-membership $t: s$ then $A, \theta_A \models^\Omega_\Omega at'$ if and only if there is some $a \in A_s$ with $a \equiv_A \theta_A(t)$ if and only if there is some $b \in B_s$ with $b \equiv_B \theta_B(t)$ (because

\[\text{since the signature is coalgebraic, it follows that disjoint unions of algebras are well defined and are exactly the coproducts in } \text{HMAAlg}_{(\Omega,D,F)}\].
e is also surjective on sorts, so we take \( b = e(a) \) if and only if \( B, \theta_B \models^\Omega_0 a' \). Therefore, by Corollary 4.5, \( \text{HMA}_{\Omega,D,F} \) has a final object.

We claim, without proof, that the conditions in the above theorem also imply that the category of hidden membership \((\Omega, D)\)-algebras satisfying the sentences in \( F \) is isomorphic to a category of coalgebras for a suitably chosen functor. Based on this intuition as well as on the fact that data is often presented loosely by axioms in practice, we introduce the following:

**Definition 4.9** Given a coalgebraic hidden membership signature \((\Omega, D)\), a fixed-data behavioral membership \((\Omega, D)\)-theory \( B = (\Omega, F) \) is called **coalgebraic** iff \( \text{HMA}_{\Omega,D,F} \) is isomorphic to a category of coalgebras with a final model. Similarly, given a loose-data hidden membership signature \( \Omega \) such that each operation has at most one argument of hidden kind, a loose-data behavioral membership \( \Omega \)-theory \( B = (\Omega, F) \) is called **coalgebraic** iff for any data algebra \( D \) satisfying \( B|_{K_V} \), \( \text{HMA}_{\Omega,D,F} \) is isomorphic to a category of coalgebras admitting a final model, where \( B|_{K_V} = (\Omega|_{K_V}, F \cap \text{Sign}_{\text{MEL}}(\Omega|_{K_V})) \).

In the rest of this paper, we will only use the fact that coalgebraic behavioral membership specifications admit a final model for any appropriate fixed-data, for which Theorem 4.8 provides a sufficient syntactic criterion. To simplify the exposition, we let \( \text{CBM}_{\text{MEL}} \) denote the institution whose signatures are the coalgebraic hidden membership signatures of Definition 4.1, whose sentences are restricted to those having at most one hidden variable like in Theorem 4.8, and whose models and satisfaction are defined like in \( \text{BM}_{\text{MEL}} \). Then notice that for \( B = (\Omega, F) \) a theory in \( \text{CBM}_{\text{MEL}} \) and \( D \) a \( B|_{K_V} \)-algebra, if \( \text{HMA}_{\Omega,D,F} \) is nonempty then it has a final object. Notice also that there is an obvious embedding of \( \text{CBM}_{\text{MEL}} \) into \( \text{BM}_{\text{MEL}} \).

5 BMaude: Language Design and Institutional Foundations

In this section we explore some fundamental aspects regarding BMaude’s module system. A major decision in our design effort is that BMaude should extend the current design of Full Maude in a smooth and practical way, both at the level of basic (flat) specifications and at the level of structured specifications.

5.1 Functional Modules and Theories: Freeness Constraints

Functional modules and theories are part of the Maude language and are supported by the Full Maude system. They are theories in the MEL sublogic of BMEL. From the BMEL point of view we will often call MEL theories **visible theories**, since they coincide with those BMEL theories for which all kinds are visible.

Note that MEL is a liberal institution [69]; that is, for each theory morphism \( \phi : T \to T' \) in \( \text{Th}_{\text{MEL}} \), the forgetful functor \( \cdot|_{\phi} : \text{MAlg}_{T'} \to \text{MAlg}_T \)
has a left adjoint \( \mathcal{F}_\phi : \mathbf{MA}_{\mathbb{T}} \to \mathbf{MA}_{\mathbb{T}'} \). This liberality is the basis for the distinction between functional modules and functional theories in Maude.

Functional modules, declared with syntax \texttt{fmod ... endfm}, satisfy \textit{freeness constraints} in the sense of [12] and therefore have an \textit{initial} or, more generally, \textit{free extension} semantics. By contrast, functional theories, declared with syntax \texttt{ftth ... endftth}, have a \textit{loose} semantics in which all models are allowed (but see below for a more precise statement).

In Maude the above theory map is a \textit{theory inclusion} \( J : T \hookrightarrow T' \) in \( \text{Th}_{\text{MEL}} \), and \( T' \) is then declared as a \textit{parameterized functional module}. The free extension semantics of such a module then means that the admissible models are those membership algebras \( A' \in \mathbf{MA}_{\mathbb{T}'} \) satisfying the freeness constraint that the counit map \( \epsilon_{A'} : \mathcal{F}_J(A' \downharpoonright J) \to A' \) is an \textit{isomorphism}. For example, the parameterized functional module,

\begin{verbatim}
  fmod LIST(X :: TRIV) is
    sort List(X) .
    op nil : -> List(X) .
    op _._ : Elt.X List(X) -> List(X) .
    op _@_ : List(X) List(X) -> List(X) .
    var E : Elt.X .
    vars L L' : List(X) .
    eq nil @ L = L .
    eq (E . L) @ L' = E . (L @ L') .
  endfm
\end{verbatim}

imposes the freeness constraint associated to the theory inclusion \( \text{TRIV}.X \hookrightarrow \text{LIST} \), where the parameter \( \text{TRIV}.X \) is a renamed copy of the \( \text{TRIV} \) theory having a single sort \( \text{Elt} \) and with no axioms. Intuitively, the admissible models have as data of sort \( \text{List} \) the lists over the data of sort \( \text{Elt}.X \). The case of \textit{unparameterized} functional modules, which have an \textit{initial} semantics – for example a module \( \text{NAT} \) defining the natural numbers in Peano notation – can be viewed as the special case of the above scheme where the theory inclusion is of the form \( J : \emptyset \hookrightarrow T' \), for \( \emptyset \) the theory with empty signature and empty axioms.

The above account simplifies things somewhat, in that in Full Maude we consider \textit{structured theories}, which are hierarchies of theory inclusions. So being a theory doesn’t mean not having any freeness constraints: it means \textit{not having them at the top}, but perhaps having some in subtheories. For example, the theory \( \text{POSET} \) of partially ordered sets has a loose semantics, but it must \textit{protect} its \( \text{BOOL} \) subtheory; that is, it has an initiality constraint in its subtheory \( \text{BOOL} \). Structured theories are further discussed in Section 5.4.

5.2 Behavioral Modules and Theories: Finality Constraints

In BMaude we can also declare \textit{behavioral modules} and \textit{behavioral theories}. Behavioral \textit{theories}, declared with syntax \texttt{bfth ... endbfth}, are theories \( \mathcal{B} \in \text{Th}_{\text{BMEL}} \) with a \textit{loose semantics}, that is, allowing any model; but this should be qualified as in Section 5.1 by noting that such theories may have constraints.
(of freeness, and now also of finality) in their subtheories, which then must be satisfied by the allowable models of $B$.

Behavioral *modules*, declared with syntax `bmod...endbm`, are required to be *coalgebraic* behavioral theories $B \in \text{Th}_{CBMEL}$. In BMaude such a module will have several visible, i.e., functional, subtheories. In fact, we adopt the general language convention that:

A sort or a kind is visible iff it is declared in a functional theory or module, and it is hidden iff it is declared in a behavioral theory or module.

Let $T$ be the union of all the visible subtheories of $B$ (which is their colimit in the subcategory of MEL theories and theory inclusions) and consider the theory inclusion, $J: T \hookrightarrow B$ in $\text{Th}_{CBMEL}$. Since $T$ is visible and $B$ coalgebraic, for each $T$-algebra $D$ the category $\text{HMAlg}_{(B,D)}$, if it is nonempty has a final object $Z_{(B,D)}$. Declaring $B \in \text{Th}_{CBMEL}$ as a behavioral module, places the *finality constraint* that the admissible models are those $A \in \text{HMAlg}_B$ such that the unique morphism $A \rightarrow Z_{(B,A \downarrow T)}$ is an isomorphism.\(^{12}\)

The visible subtheories of $B$ may or may not have freeness constraints; that is, they may be either functional modules, or functional theories. The behavioral module $B$ will then be *parameterized* by those visible subtheories that do not have freeness constraints at their top. For example, the following parameterized behavioral module defines streams for any parameter set of data elements.

```
bfmod STREAM(X :: TRIV) is
  sort Stream(X) .
  op head : Stream(X) \rightarrow Elt.X .
  op tail : Stream(X) \rightarrow Stream(X) .
  op _&_ : Elt.X Stream(X) \rightarrow Stream(X) .
  var E : Elt.X . var S : Stream(X) .
  eq head(E & S) = E .
  eq tail(E & S) = S .
endbfm
```

5.3 The Institution $C_2^F(BMEL)$

We can begin to place BMaude’s modules and theories in an institutional setting by considering a logic extension of behavioral membership equational logic (BMEL) in which freeness and finality constraints are viewed as additional sentences in the extended logic. We will denote this logic extension by $C_2^F(BMEL)$. Our approach generalizes the treatment of freeness constraints in \(^{12}\). Note that we have a full subcategory inclusion $\text{Sign}_{MEL} \hookrightarrow \text{Sign}_{BMEL}$ embedding visible (MEL) signatures into the category of BMEL-...
signatures. Note that we also have subcategory inclusions at the level of theories, \( \text{Th}_{\text{MEL}} \hookrightarrow \text{Th}_{\text{CBMEL}} \hookrightarrow \text{Th}_{\text{BMEL}} \), embedding visible theories into coalgebraic behavioral theories, and these into general behavioral theories. In what follows we will treat all the above embeddings for both signatures and theories as actual inclusions.

For \( \Omega \in \text{Sign}_{\text{BMEL}} \) we define an \( \Omega \)-freeness constraint as a pair \((J,G)\), where \( J: T \hookrightarrow T' \) is a theory inclusion in \( \text{Th}_{\text{MEL}} \), and \( G: \text{sign}(T') \to \Omega \) a signature morphism in \( \text{Sign}_{\text{BMEL}} \), where \( \text{sign}(T') \) denotes the underlying signature of the theory \( T' \). We denote by \( \text{Sen}_\mathcal{F} (\Omega) \) the set\(^\text{13}\) of all \( \Omega \)-freeness constraints.

A hidden membership \( \Omega \)-algebra \( A \) satisfies the freeness constraint \((J,G)\), denoted \( A \models_\mathcal{F} (J,G) \), if and only if the counit map \( \epsilon_{A|G} : \mathcal{F}_J((A \mid G)\mid_J) \to A|G \) is an isomorphism. Note that a signature morphism \( K: \Omega \to \Delta \) induces a function \( \text{Sen}_\mathcal{F}(K): \text{Sen}_\mathcal{F}(\Omega) \to \text{Sen}_\mathcal{F}(\Delta) \) from \( \Omega \)-freeness constraints to \( \Delta \)-freeness constraints by the rule, \( \text{Sen}_\mathcal{F}(K)(J,G) = (J,G;K) \). This defines a functor \( \text{Sen}_\mathcal{F}: \text{Sign}_{\text{BMEL}} \to \text{Set} \). Note also that, by the functoriality of the functor \( \text{Mod}_{\text{BMEL}}: \text{Sign}_{\text{BMEL}} \to \text{Cat}^{\text{op}} \), the following satisfaction condition holds for any hidden membership \( \Delta \)-algebra \( B \):

\[
B|_K \models_\mathcal{F} (J,G) \iff B \models_\mathcal{F} \text{Sen}_\mathcal{F}(J,G).
\]

Therefore, \((\text{Sign}_{\text{BMEL}}, \text{Mod}_{\text{BMEL}}, \text{Sen}_\mathcal{F}, \models_\mathcal{F})\) is an institution.

Finality constraints define another institution in a completely similar way. For \( \Omega \in \text{Sign}_{\text{BMEL}} \) we define an \( \Omega \)-finality constraint as a pair \((J,G)\), where \( J: T \hookrightarrow B \) is a theory inclusion with \( T \in \text{Th}_{\text{MEL}} \) and \( B \in \text{Th}_{\text{CBMEL}} \), and where \( G: \text{sign}(B) \to \Omega \) is a signature morphism in \( \text{Sign}_{\text{BMEL}} \). We denote by \( \text{Sen}_\mathcal{Z}(\Omega) \) the set of all \( \Omega \)-finality constraints.

A hidden membership \( \Omega \)-algebra \( A \) satisfies the finality constraint \((J,G)\), denoted \( A \models_\mathcal{Z} (J,G) \), if and only if the unique morphism \( A|G \to \mathcal{Z}_{(B,(A|G)\mid_J))} \) is an isomorphism. Note that a signature morphism \( K: \Omega \to \Delta \) induces a function \( \text{Sen}_\mathcal{Z}(K): \text{Sen}_\mathcal{Z}(\Omega) \to \text{Sen}_\mathcal{Z}(\Delta) \) from \( \Omega \)-finality constraints to \( \Delta \)-finality constraints by the rule, \( \text{Sen}_\mathcal{Z}(K)(J,G) = (J,G;K) \). This defines a functor \( \text{Sen}_\mathcal{Z}: \text{Sign}_{\text{BMEL}} \to \text{Set} \). Note also that, by the functoriality of the functor \( \text{Mod}_{\text{BMEL}}: \text{Sign}_{\text{BMEL}} \to \text{Cat}^{\text{op}} \), the following satisfaction condition holds for any hidden membership \( \Delta \)-algebra \( B \):

\[
B|_K \models_\mathcal{Z} (J,G) \iff B \models_\mathcal{Z} \text{Sen}_\mathcal{Z}(J,G).
\]

Again, this makes \((\text{Sign}_{\text{BMEL}}, \text{Mod}_{\text{BMEL}}, \text{Sen}_\mathcal{Z}, \models_\mathcal{Z})\) into an institution. Our desired institution \( \mathcal{C}_\mathcal{Z}(\text{BMEL}) \) is, by definition, the institution \( \mathcal{C}_\mathcal{Z}(\text{BMEL}) = (\text{Sign}_{\text{BMEL}}, \text{Mod}_{\text{BMEL}}, \text{Sen}_{\mathcal{BMEL}} \uplus \text{Sen}_\mathcal{F} \uplus \text{Sen}_\mathcal{Z}, \models_\mathcal{Z}) \), where \( \uplus \) denotes disjoint union, and where \( \models_\mathcal{Z} \) coincides, for the sentences of each of the three

\(^{13}\)We ignore foundational issues about the size of this set by assuming some universe of sets to which all theories, and theory and signature morphisms, belong as elements.
institutions involved, with the satisfaction relation in that institution, that is, either $\models$, or $\models_F$, or $\models_Z$. The interested reader is referred to [87,52] for more on colimits of institutions. Note that, by a general result in [42], since $\text{Sign}_{\text{BMEL}}$ is cocomplete, the category of theories $\text{Th}_{c^Z_{BMEL}}$ is also cocomplete.

5.4 Structured Theories: The Institution $S(C^F_Z(\text{BMEL}))$

We are not yet done since, as already mentioned, BMAude modules are structured theories; that is, they are hierarchies of theory inclusions, so that a theory has a collection of subtheories, which may share other subtheories below, and so on. We summarize and apply here the general method of Durán and Meseguer [37] that constructs out of any institution $\mathcal{I}$ another institution $S(\mathcal{I})$ whose ordinary theories are the structured $\mathcal{I}$-theories. This method has already been applied to endow Full Maude with a categorico-institutional semantics for its module algebra [36,32,38]. Here we extend those ideas to BMAude.

First of all, we must view theory and signature hierarchies as diagrams, where in our actual uses the diagram schema will be a finite poset, and the image by the diagram of a pair $i \leq j$ in such a poset will be an inclusion. This is a special case of the following general definition:

**Definition 5.1** Let $\mathcal{C}$ be a category. The **diagram category** $\text{Dgm}(\mathcal{C})$ has as objects functors $\mathcal{D}: \mathcal{P} \to \mathcal{C}$, where $\mathcal{P}$ is a small category. If $\mathcal{D}: \mathcal{P} \to \mathcal{C}$ and $\mathcal{D}': \mathcal{P}' \to \mathcal{C}$ are objects, then a morphism $(R, \rho): \mathcal{D} \to \mathcal{D}'$ consists of a functor $R: \mathcal{P} \to \mathcal{P}'$ and a natural transformation $\rho: \mathcal{D} \Rightarrow R; \mathcal{D}'$. The composition of morphisms $(R, \rho)$ and $(R', \rho')$, as depicted in the diagram below, is given by the morphism $(R; R', \rho \cdot (1_R; \rho'))$.

![Diagram](image)

where $\cdot$ and $\Rightarrow$ denote the vertical and the horizontal compositions of natural transformations, respectively.

A **structured signature** can be formalized as a functor $\mathcal{D}: \mathcal{I} \to \text{Sign}_{\mathcal{I}}$ from a small category $\mathcal{I}$ to the category $\text{Sign}_{\mathcal{I}}$ of signatures and signature morphisms in a given institution $\mathcal{I}$. Durán and Meseguer [37] define an institution $S(\mathcal{I})$, whose theories are called structured $\mathcal{I}$-theories, by defining functors $\text{Sen}_{S(\mathcal{I})}$ and $\text{Mod}_{S(\mathcal{I})}$ associating to each structured signature $\mathcal{D}$ in $\text{Sign}_{S(\mathcal{I})}$ a set of $\mathcal{D}$-sentences and a category of $\mathcal{D}$-models, respectively. The definitions are as follows:
Definition 5.2 [37] Let us denote by $\text{Sign}_S(I)$ the category $\text{Dgm} (\text{Sign}_S)$ of diagrams over the category of signatures in the institution $I$. We shall call the objects of $\text{Sign}_S(I)$ structured $(I,\mathcal{S})$-signatures, and will denote each structured signature by its corresponding diagram $\mathcal{D} : I \to \text{Sign}_S$. The morphisms in $\text{Sign}_S(I)$ are called structured signature morphisms.

Definition 5.3 [37] The functor $\text{Sen}_S(I) : \text{Sign}_S(I) \to \text{Set}$, associating to each structured signature $\mathcal{D} : I \to \text{Sign}_S$ a set of sentences and to each structured signature morphism $(K, H) : \mathcal{D} \to \mathcal{D}'$ a corresponding translation at the level of sentences, is defined as follows:

$$\text{Sen}_S(I)(\mathcal{D}) = \coprod_{i \in I} \text{Sen}_S(\mathcal{D}(i))$$

$$\text{Sen}_S(I)((K, H)) = \coprod_{i \in I} \text{Sen}_S(H_i)$$

We can see each of the sentences of $\mathcal{D}$ as a pair $(i, f)$, where $f \in \text{Sen}_S(I)(\mathcal{D}(i))$.

Definition 5.4 [37] Given a structured signature $\mathcal{D} : I \to \text{Sign}_S$, its category of models $\text{Mod}_S(I)(\mathcal{D})$ has as objects families $M = \{M_i\}_{i \in I}$ with $M_i$ in $\text{Mod}_S(I)(\mathcal{D}(i))$, such that for each $\alpha : i \to j$ in $I$, $\text{Mod}_S(I)(\mathcal{D}(\alpha))(M_j) = M_i$. A morphism between two such models $f : M \to M'$ is given by a family $\{f_i : M_i \to M'_i\}_{i \in I}$ with $f_i$ in $\text{Mod}_S(I)(\mathcal{D}(i))$ such that for each $\alpha : i \to j$ in $I$, $\text{Mod}_S(I)(\mathcal{D}(\alpha))(f_j) = f_i$.

Definition 5.5 [37] The functor $\text{Mod}_S(I) : \text{Sign}_S(I) \to \text{Cat}^{\text{op}}$ assigns to each structured signature $\mathcal{D} : I \to \text{Sign}_S$ its category of models $\text{Mod}_S(I)(\mathcal{D})$, and to each structured signature morphism $(K, H) : \mathcal{D} \to \mathcal{D}'$ the forgetful functor $\text{Mod}_S(I)((K, H)) : \text{Mod}_S(I)(\mathcal{D}') \to \text{Mod}_S(I)(\mathcal{D})$, defined as follows:

$$\text{Mod}_S(I)((K, H))((M'_j)_{j \in I'}) = \{\text{Mod}_S(I)(H_i)(M'_{K(i)})\}_{i \in I}$$

$$\text{Mod}_S(I)((K, H))((f'_{j})_{j \in I'}) = \{\text{Mod}_S(I)(H_i)(f'_{K(i)})\}_{i \in I}$$

Definition 5.6 [37] Given a structured signature $\mathcal{D} : I \to \text{Sign}_S$, a $\mathcal{D}$-model $M = \{M_i\}_{i \in I}$ satisfies a $\mathcal{D}$-sentence $(i, f)$ if and only if $M_i \models_{\mathcal{D}(i)} f$. In this case, we write $M \models_{\mathcal{D}} (i, f)$.

Definition 5.7 [37] Let $S(I)$ be the institution with:

- $\text{Sign}_S(I)$ as category of signatures,
- the sentence functor $\text{Sen}_S(I) : \text{Sign}_S(I) \to \text{Set}$, of Definition 5.3
- the model functor $\text{Mod}_S(I) : \text{Sign}_S(I) \to \text{Cat}^{\text{op}}$, of Definition 5.5, and
- the satisfaction relation given in Definition 5.6 for which the satisfaction condition holds as shown in [37] Proposition 13].

Note that the notion of structured $I$-theory, that is, of a theory presentation in $S(I)$, captures well the intuitive notion of structured theory found in
actual specifications. Indeed, when a subtheory is imported, its axioms typically are not repeated; they are implicitly inherited from the subtheory. This means that axioms are presented locally, for a specific local signature $D(i)$, corresponding to the above formal notion of a pair $(i, f)$. It also means that at each stage in the specification only the incremental information of additional axioms has to be made explicit. This correspondence with the actual specification practice can be made even more intuitive by remarking that, in practice, theories are typically named entities like $\text{bool}$, $\text{nat}$, $\text{list}$, etc. Therefore, we should think of the index set $|I|$ in a structured signature $D: I \rightarrow \text{Sign}_{\iota}$ as the set of names for the different theories present in the structure. Then a sentence such as $\text{null} . L = L$, say the theory $\text{list}$, is expressed in the above framework as the pair $(\text{list}, \text{null} . L = L)$, indicating how the axiom is localized to the $\text{list}$ component of the corresponding structured theory.

Since $\text{Sign}_{\iota(I)} = \text{Dgm}(\text{Sign}_{\iota})$, there is a close and systematic relationship between the category $\text{Th}_{\iota(I)}$ of structured $\iota$-theories in the institution $\iota$ and the diagram category $\text{Dgm}(\text{Th}_{\iota})$. This relationship can be expressed as an adjunction. Let $J : \text{Th}_{\iota(I)} \rightarrow \text{Dgm}(\text{Th}_{\iota})$ be the functor defined on objects as $J(D, F) = D_{F^*}$, where if $D : I \rightarrow \text{Sign}_{\iota}$ is a structured signature, then $D_{F^*} : I \rightarrow \text{Th}_{\iota}$ has $D_{F^*}(i) = (D(i), F_i^*)$ and $D_{F^*}(\alpha : i \rightarrow j) = D(\alpha)$, where $F_i^* = \{ f \in \text{Sen}_{\iota}(D(i)) \mid \forall M \in \text{Mod}_{\iota}(D, F), M_i \models_{D(i)} f \}$. The definition of $J$ on morphisms assigns to each theory morphism $(K, H) : (D, F) \rightarrow (D', F')$ in $\text{Th}_{\iota(I)}$ the diagram morphism $(K, \tilde{H}) : D_{F^*} \rightarrow D'_{F'^*}$, with $\tilde{H}_i = H_i$ for each $i \in I$.

**Proposition 5.8** [37] The functor $J : \text{Th}_{\iota(I)} \rightarrow \text{Dgm}(\text{Th}_{\iota})$ is full and faithful, and has a left adjoint $R$.

Several key results about the cocompleteness of the categories of structured signatures and theories are given in [37]. One of these results states that, if the category of signatures of $\iota$ has colimits, then the categories of signatures and theories of $\iota$ both have colimits, making then possible extending the proposal of Goguen and Burstall [42] of taking colimits of theories as a systematic way of “putting theories together” to structured theories. That is, one can use colimits of structured theories as a systematic way of putting structured theories together. In particular, the semantics of the instantiation of structured theories is given by the pushouts in the category of structured theories, which can be obtained, using the functor $J$, from pushouts in $\text{Dgm}(\text{Th}_{\iota})$.

In BMaude we need more, namely structured theories with freeness and finality constraints among their sentences. But this is now very easy, since these are theories in the institution $\iota(\mathcal{C}_F^E(\text{BMEL}))$. Note that since the category of signatures of $\mathcal{C}_F^E(\text{BMEL})$ has colimits, the category of theories for $\iota(\mathcal{C}_F^E(\text{BMEL}))$ also has colimits. This supports a “module algebra” for structured theories entirely similar to that of Full Maude [38]. In particular, both modules and theories can be parameterized, and both can be instantiated by parameterized views, that is, by parameterized and structured theory mor-
We refer to [38] for both syntactic conventions and categorical semantics of parameterized theories, modules, and views, which carry over naturally to BMaude. Here we just illustrate the main concepts introduced in this section by discussing a BMaude example and how it can be naturally formalized as a structured theory in $\mathcal{S}(C^F_Z(\text{BMEL}))$. Note that there is an obvious theory morphism (view) $\text{Ring}$ mapping the $\text{TRIV}$ theory to the theory $\text{RING}$ of (commutative) rings. We can then define dataflow addition and multiplication operations on streams whose elements belong to a ring by first instantiating the $\text{TRIV}$ parameter of $\text{STREAM}$ to $\text{RING}$, thus getting streams over a ring; and then extending streams over a ring with the new dataflow operations in a parameterized behavioral theory extending streams over a ring.

```plaintext
bfth DATAFLOW(R :: RING) is
  protecting STREAM(Ring)(R) .
  op _+_ : Stream(Ring)(R) Stream(Ring)(R) -> Stream(Ring)(R) .
  op _*_ : Stream(Ring)(R) Stream(Ring)(R) -> Stream(Ring)(R) .
  vars S S' : Stream(Ring)(R) .
  eq head(S + S') = head(S) + head(S') .
  eq tail(S + S') = tail(S) + tail(S') .
  eq head(S * S') = head(S) * head(S') .
  eq tail(S * S') = tail(S) * tail(S') .
endbfth
```

Note the protecting declaration, stating that the finality constraint for the $\text{STREAM}$ module is preserved by the importation. We can then instantiate this parameterized theory to perform dataflow operations on streams of integers by means of a view $\text{INT-as-R}$ from $\text{RING}$ to $\text{INT}$, thus getting dataflow operations on integer data:

```plaintext
bfth INT-DATAFLOW is
  including DATAFLOW[INT-as-R] .
endbfth
```

Since $\text{DATAFLOW}$ is a theory and not a module, no constraints have to be preserved at the top level; therefore the including keyword is used. However, as in Full Maude we follow the convention that:

All freeness and finality constraints in (proper) subtheories of a theory imported in including mode are preserved, unless explicitly stated otherwise by importing some of those subtheories also in including mode.

The structured theory with freeness and finality constraints corresponding to the behavioral theory $\text{INT-DATAFLOW}$ is depicted below, where we have indicated the theory inclusion having a freeness constraint by $\Rightarrow$, and that having a finality constraint by a triple arrow:
6 BMaude: Reflective Architecture

In this section we explain how the entire BMaude extension of Maude can be implemented in Maude using reflection, without any need to alter or extend Maude’s underlying C++ implementation. This means that a BMaude implementation can be obtained with much less effort, and in a much more flexible and extensible way, than would be possible with a conventional implementation.

The situation is exactly the same as with Full Maude’s reflective architecture [36,32], which is entirely written in Maude and was indeed developed relatively quickly and with moderate effort. In fact, things are now even better, because BMaude will itself extend Full Maude and will inherit large parts of its functionality. For example, all Full Maude module operations for functional modules and theories will be inherited by BMaude without change, and will only require modest changes when extended to behavioral modules and theories.

6.1 Rewriting Logic Reflection and Maude’s META-LEVEL

BMaude’s reflective architecture uses crucially the fact that rewriting logic [68] is reflective [19,23] in the precise sense that there is a finitely presented rewrite theory $U$ which is universal, that is, for any finitely presented rewrite theory $R$ (including $U$ itself) we have the following equivalence:

$$R \vdash t \rightarrow t' \iff U \vdash \langle R, t \rangle \rightarrow \langle R, t' \rangle,$$

where $R$, $t$, and $t'$ are terms representing, respectively, $R$, $t$, and $t'$ as data elements of $U$.

Maude’s design and implementation systematically exploit the reflective capabilities of rewriting logic, providing key features of the universal theory $U$ in its builtin module META-LEVEL [21]. In particular, META-LEVEL has sorts Term
and Module, so that the representations of a term \( t \) and of a module \( R \) are, respectively, a term \( T \) of sort Term and a term \( R \) of sort Module.

The module \texttt{META-LEVEL} also provides key metalevel functions for matching, rewriting and evaluating terms at the metalevel, including: \texttt{metaXmatch} (match a pattern term to a subterm of a subject term), \texttt{metaApply} (apply a rule at the top), \texttt{metaXApply} (apply a rule at some subterm position), \texttt{metaRewrite} (do default interpreter rewriting), and \texttt{metaReduce} (apply only equations, to get a canonical form). \texttt{META-LEVEL} has also generic parsing and pretty printing functions \texttt{metaParse} and \texttt{metaPrettyPrint} that, respectively, parse an input string according to the grammar of a given module of sort Module, and pretty print a term of sort Term according to the mix-fix grammar and lexical conventions of its corresponding module of sort Module.

6.2 BMaude’s Reflective Architecture

We sketch here the key functionality of the planned BMaude reflective implementation as an extension of that of Full Maude, which itself extends \texttt{META-LEVEL}. There are three main areas to consider:

1. module algebra;
2. input and output; and
3. support for expression evaluation and deduction.

We discuss below areas (1)–(2). Area (3) is discussed in detail in Section 7. By a module algebra we mean all the composition operations allowing us to combine, reuse, and transform different structured modules and theories. As in Full Maude, the idea is to represent in Maude BMaude’s modules and theories with their hierarchical structure, including their parameters if they are parameterized. The institutional foundations of such structured modules and theories have already been discussed in Section 5.4. In their reflective implementation the idea is to metarepresent structured modules and theories as terms in sorts extending the sort Module of unstructured modules in \texttt{META-LEVEL}. Among such sorts there will be sorts StrTheory and StrModule, both unified in a sort Unit containing them as subsorts. We also need a sort View to metarepresent theory morphisms between structured theories and modules. As in Full Maude \cite{36,32}, the idea is that the semantics of each module operation can be defined by confluent and terminating equations at the metalevel, so that the resulting canonical form after such module expressions are evaluated is a structured module. Reflection has the important advantage of making the module algebra easily extensible by new operations, as illustrated for Full Maude in \cite{32,34}.

Of course, we do not only want to apply some module transformation on some particular input; we also want to be able to interact with the system, entering modules, theories and views to which we can then refer by name in other module expressions. For this purpose we want to be able to store
modules, theories, and views in a database, so that they can be referred to later in order to evaluate module expressions, evaluate commands, and so on. This means that there is another data type of sort Database, representing in fact a kind of object, and containing the different structured theories, modules, and views defined so far. The semantics of the database can then be defined in an object-oriented way by means of rewrite rules that specify how the database is queried and updated.

The BMaude database with which the user interacts leads us to the second area of functionality, namely parsing, pretty printing, and input/output. In Maude, these additional metalanguage features are supported in a reflective way as follows: the BMaude syntax definition is accomplished by defining a data type grammarBMaude, specified by a Maude functional module whose signature specifies the grammar of BMaude; particularities at the lexical level can be accommodated by user-definable bubble sorts \cite{20}, that tailor the adequate notions of token and identifier to the specific syntax needs of BMaude; parsing and pretty printing of BMaude modules, expressions, and commands is then accomplished by the predefined functions metaParse and metaPrettyPrint in META-LEVEL, in conjunction with the bubble sorts defined in the BMaude signature. Input/output of BMaude module and theory definitions, and of commands for execution is then accomplished by means of rewrite rules specifying both the effect of those commands in the Database and the interactions by means of messages between the Database object and other builtin Maude objects, such as files and graphical user interfaces, that will be provided by the upcoming Maude 2.0 release to support different forms of interaction with the user.

7 BMaude: Behavioral Reasoning

We next discuss possible implementations of automated behavioral reasoning in BMaude. The main techniques that we consider are behavioral rewriting and its combination with coinduction.

7.1 Behavioral Rewriting

Behavioral rewriting provides for the behavioral equational inference rules presented in Subsection 3.3 what standard term rewriting does for equational logic.

Definition 7.1 An Ω-rewriting rule\footnote{We only discuss behavioral unconditional rewriting here. Rewriting modulo axioms is not discussed here either, but we hope to address these generalizations elsewhere soon.} is a triple \((∀Y) l → r\), where \(l, r ∈ T_Ω(Y)\). A behavioral rewriting system is a triple \(R = (Ω, Γ, R)\), where \(Ω\) is a hidden membership signature, \(Γ\) is a hidden membership subsignature of \(Ω\), and \(R\) is a set of \(Ω\)-rewriting rules.

\[43\]
Ordinary term rewriting is \emph{not sound} for behavioral satisfaction. This is because non-behavioral operations may not preserve the behavioral equivalence relation. Consequently, term rewriting needs to be modified accordingly. From now on in this subsection we suppose that \( R = (\Omega, \Gamma, R) \) is a behavioral \( \Omega \)-rewriting system and \( B = (\Omega, \Gamma, E) \) is the associated behavioral specification, that is, \( E = \{ (\forall Y) \ l = r \mid (\forall Y) \ l \rightarrow r \in R \} \).

**Definition 7.2** The \textbf{behavioral (term) rewriting relation} associated to the behavioral rewriting system \( R \) is the smallest relation \( \Rightarrow \) such that:

1. for each \( (\forall Y) \ l \rightarrow r \in R \) and each \( \theta : Y \rightarrow T_\Omega(X) \), \( \theta(l) \Rightarrow \theta(r) \),
2. if \( t \Rightarrow t' \) and \( \text{kind}(t, t') \in K_V \) then \( \sigma(W, t) \Rightarrow \sigma(W, t') \) for all \( \sigma \in \text{Der}(\Omega) \),
3. if \( t \Rightarrow t' \) and \( \text{kind}(t, t') \in K_H \) then \( \delta(W, t) \Rightarrow \delta(W, t') \) for all \( \delta \in \Gamma \).

When \( R \) is important, we write \( \Rightarrow_R \) instead of \( \Rightarrow \).

Behavioral rewriting modifies standard term rewriting as follows: each time a hidden redex is found, the rewriting rule is enabled only if all operations on the path from that redex going toward the root until a visible sort is found are behavioral (or behaviorally congruent). If a visible sort is not found, the rewriting is still enabled if all operations on the path from the redex to the root are behavioral.

**Definition 7.3** An \( \Omega \)-context \( c \) is \textbf{behavioral} iff all operations on the path to \( \ast \) in \( c \) are behavioral, and \( c \) is \textbf{safe} iff either it is behavioral or there is some behavioral experiment \( c' \) such that \( c = c''[c'] \) for some appropriate \( c'' \).

The following can be seen as an equivalent definition of behavioral rewriting:

**Proposition 7.4** \( t \Rightarrow t' \) iff there is a rewriting rule \( (\forall Y) \ l \rightarrow r \in R \), a safe context \( c \), and a substitution \( \theta \) such that \( t = c[\theta(l)] \) and \( t' = c[\theta(r)] \).

CafeOBJ \[28\] and BOBJ \[45\] both implement behavioral rewriting internally, as part of their core rewriting engines. By contrast, very fast and elegant rewriting engines such as Maude \[20\] and Elan \[11\] do not currently support behavioral rewriting. Adding behavioral rewriting to those engines, though possible, may require nontrivial extensions. However, using Maude’s reflective capabilities it is possible to implement behavioral rewriting in a quite easy way, without any change to the core rewrite engine. We present two such reflective designs below.

The Maude \texttt{META-LEVEL} module \[20,21\] provides, among other features, a sort \texttt{Module} for special terms that represent modules, a sort \texttt{Term} for representations of ordinary terms, and an operation \texttt{metaReduce} taking a module \( M \) and a term \( t \) and returning the normal form of \( t \) in \( M \). What one needs to do is to extend \texttt{META-LEVEL} with a new sort for behavioral modules or theories, say \texttt{Unit}, together with appropriate constructors for their syntax, and also to add a new meta-reduce operation, say \texttt{metaBReduce}, taking a behavioral module or theory \( B \) and a term \( t \) and returning the behavioral normal form.
of \( t \) in \( B \). The major part of such a meta-level implementation for behavioral rewriting is to find an appropriate implementation of \texttt{metaBReduce}. We discuss two different ways to do so below.

One implementation of \texttt{metaBReduce} would just follow the equivalent definition of behavioral rewriting in Proposition 7.4, using the \texttt{metaXmatch} operation provided by the new Maude 2.0 META-LEVEL [21], which takes a module and two terms and returns possible matchings of the two terms, at any positions; a match consists of a substitution together with a \textit{context}, where the match can take place. What is left to be done now is to implement a predicate that tests if the context returned by \texttt{metaXmatch} is \textit{safe} in the signature of the given behavioral module (see Definition 7.3), which should be straightforward, and if this is the case then to execute the rewriting step using \texttt{metaXapply}, also provided by the new Maude 2.0 META-LEVEL [21].

We next describe another possible technique to implement \texttt{metaBReduce}, based on coloring the operations in terms with either red or green, with the intuition that a rewriting step is allowed only if its redex has a green root. A canonical rewriting system maintaining the consistency of colorings can be devised, called the \textit{painter}. This technique was first presented in [44] without proofs, and is explored in more detail in [90,88].

Let us imagine that operations are painted with either green or red. To keep the notation simple, we just add a subscript “\( r \)” to the red copies of some operations. More precisely, given a hidden membership signature \( \Omega \), let \( \Omega_r \) be the standard membership signature having the same sorts as \( \Omega \) and an operation \( \sigma_r : k_1...k_n \rightarrow k_h \) for each operation of hidden result \( \sigma \) in \( \Omega \), and let \( \Omega' \) be \( \Omega \cup \Omega_r \cup \{ g : k \rightarrow k \mid \text{for all kinds } k \} \). For a hidden subsignature \( \Gamma \) of \( \Omega \), let \( \text{Painter}_{\Omega, \Gamma} \) be the standard \( \Omega' \)-term rewriting system:

1. \( (\forall X) \ g(\sigma_r(X)) \rightarrow g(\sigma(X)) \)
2. \( (\forall Z,X) \ \tau(Z,\sigma_r(X)) \rightarrow \tau(Z,\sigma(X)), \text{ for each } \tau \in ((\Omega - \Gamma) \cup \Omega_r)_{wk_h,k} \)
3. \( (\forall Z,X) \ \delta(Z,\sigma_r(X)) \rightarrow \delta(Z,\sigma(X)), \text{ for each } \delta \in \Gamma_{wk_h,k} \)

where \( \sigma \) is an operation in \( \Omega \) of hidden kind result \( k_h \). The role of these rewriting rules is to paint the terms dynamically, with the intuition that a rewriting step is allowed only on green positions. The operations \( g \) are introduced only to have control, via rules of type (1), over the operations on the top of terms which should normally be green. The second rule says that an operation of hidden kind immediately below a non-behavioral or a red operation must be red, and the third rule shows how green is propagated downwards. As shown in [90,88], \( \text{Painter}_{\Omega, \Gamma} \) is confluent and terminating.

\textbf{Example 7.5} For the behavioral rewriting system \( R = (\Omega, \Gamma, R) \) associated to the behavioral specification of nondeterministic stacks in Example 3.12, \( \text{Painter}_{\Omega, \Gamma} \) adds to \( \Omega \) the operations \( g, \text{pop}_r, \text{push}_r : NdStack \rightarrow NdStack \) and \( g : Nat \rightarrow Nat \), and the rules

\( (\forall s : NdStack) \ g(p_r(s)) \rightarrow g(p(s)) \)
\( (\forall s : NdStack) \ \text{push}(p(s)) \rightarrow \text{push}(p_r(s)) \)
(∀ s: NdStack) pop_r(p(s)) → pop_r(p_r(s))
(∀ s: NdStack) top(p_r(s)) → top(p(s))
(∀ s: NdStack) pop(p_r(s)) → pop(p(s)),

one for each p ∈ \{pop, push\}. Then the normal forms in Painter_Ω,Γ of the two Ω-terms push(pop(push(s))) and pop(pop(push(push(pop(push(s)))))) are the (Ω ∪ Ω_r)-terms push(pop_r(push_r(s))) and pop(pop(push_r(pop_r(push_r(s))))), respectively.

We let ϕ(u) denote the unique normal form in Painter_Ω,Γ of an Ω′-term u; if u is an Ω-term, ϕ(u) is called the coloring of u. Given an (Ω ∪ Ω_r)-term u, ψ(u) is the Ω-term forgetting all the colors of u, called the uncoloring of u. For the behavioral rewriting system \( R = (Ω, Γ, R) \), let ϕ(R) denote the set \( \{ (∀ Y) \varphi(l) → \varphi(r) \mid (∀ Y) l → r ∈ R \} \) of colored rules, and ϕ(Ω) denote the associated (Ω ∪ Ω_r)-rewriting system. The correctness of the following algorithm implementing behavioral rewriting by standard term rewriting is shown in [90,88]:

```
Algorithm BRE (R,t)
(i) generate ϕ(Ω) and Painter_Ω,Γ
(ii) get ϕ(t) using Painter_Ω,Γ
(iii) get normal form of g(ϕ(t)), say g(u), using ϕ(Ω) ∪ Painter_Ω,Γ
(iv) return ψ(u).
```

Step 1 can be implemented in \( O(n^2) \), where n is the size of \( R \), and it should be implemented only once for each behavioral rewriting system \( R \).

Another implementation of metaReduce in BMaude could just follow the algorithm BRE above. This implementation is somehow simpler because one does not have to implement the safety predicate on contexts. All one has to do is to generate Painter_Ω,Γ and ϕ(Ω), and then call metaReduce twice, once for the initial coloring and the other for the subsequent innermost reduction. Due to memoization options provided by the system, Painter_Ω,Γ and ϕ(Ω) would be generated only once for each behavioral rewriting system \( R \), so the overhead for one behavioral rewriting step should be, on average, the two calls to metaReduce plus the additional overhead due to coloring. Note that all the theory transformations involved, as well as the coloring and uncoloring transformations on terms can be easily defined by equations involving data elements of sorts Module, Unit, and Term in an extension of META-LEVEL. Therefore, a quite straightforward and easily extensible reflective implementation in Maude of this second design is also possible.

It is not clear to us at this stage which of the two implementations above would perform better in practice. Experimentation is certainly needed. The second one has the advantage of being less dependent on reflective primitives, so that it could be implemented as a theory transformation in other languages.

\[\text{Notice that } R \text{ is required to be weakly left linear in [90,88].}\]
provided some nonreflective implementation of such a transformation is developed. Additionally, if the initial behavioral specification is unconditional then the transformed theory is also unconditional (note that the first design requires conditional rules for safety checks); this can have a significant impact on efficiency. On the other hand, as shown in [88], the second design has some applicability limitations (e.g., the original behavioral rewriting system has to be weakly left linear) and more work is needed in order to adapt it to conditional behavioral rules; the first design does not have these limitations.

7.2 Circular Coinductive Rewriting

Our experience with BOBJ is that very few behavioral properties can be proved with just equational behavioral reasoning, so some form of coinduction was needed to be implemented. Circular coinductive rewriting integrates behavioral rewriting with circular coinduction. Its input is a pair of terms, and it returns true when it proves the terms behaviorally equivalent, and otherwise returns false or fails to terminate, much as with proving term equality by rewriting. See [88] for the (non-trivial) correctness proof; here we just describe the algorithm which was implemented in the core behavioral engine of BOBJ (slightly adapted to BMEL).

Given a behavioral specification \( B = (\Omega, \Gamma, F) \) and a cobasis \( \Delta \subseteq \Gamma \), a set of pairs of terms (which we may reorder to reduce the possibility of non-termination) \( C \), and an \( \Omega \)-term \( u \), let \( bnf_C(u) \) denote the term derived from \( u \) by rewriting as much as possible with \( F \) under the usual restrictions for behavioral rewriting, and then applying equations in \( C \) at a term position if all (zero or more) operations on the path to that position are in \( \Gamma - \Delta \).

Given a pair of \( \Omega \)-terms \( (t, t') \), the circular coinductive rewriting algorithm, hereafter denoted \( CCRW \), is as follows:

1. let \( C = \emptyset \) and \( G = \{(t, t')\} \)
2. for each \( (u, u') \) in \( G \)
3. move \( (u, u') \) from \( G \) to \( C \)
4. for each \( \delta \in \Delta \)
5. let \( v = bnf_C(\delta[u, x]) \) and \( v' = bnf_C(\delta[u', x]) \)
6. if \( v \neq v' \) then add \( (v, v') \) to \( G \)

\( G \) contains the still unproved goals and \( C \) contains the “circularities” to be used in proofs. The algorithm may fail to terminate.

This algorithm can be relatively easily implemented using the reflective capabilities of Maude. One subtle aspect of it is that the newly “discovered” equations in \( C \) need to be oriented in order to be used as rewriting rules. A default ordering is implemented in BOBJ which seems to suffice in many situations, but in BMaude we’d like to experiment with more orderings, perhaps even allow an ordering as a parameter to the CCRW procedure. Some order-
ings are explored in [35] in the context of Knuth-Bendix completion; we would like to experiment with those and see how they perform in our context.

8 Conclusion and Future Work

We have argued that further progress on the hidden logics approach can lead to a fuller semantic integration of equational, behavioral, coalgebraic, object-oriented, and rewriting logic views of systems, including concurrent and distributed ones, and can be supported by a powerful combination of formal methods to gain high assurance about those systems. Progress on integration at the foundational levels of logics and institutions should go hand in hand with advances on specification language design. Building on the BOBJ and CafeOBJ experience we have begun the design of BMaude, a behavioral extension of Maude. This paper has focused on extending behavioral logic to the more expressive framework of membership equational logic (MEL), resulting on the BMEL logic extension. We have also investigated conditions under which the models of a BMEL-specification form a category of coalgebras and have a final coalgebra. The language design and institutional foundations of BMaude for the MEL and BMEL facets have been investigated; and a reflective architecture allowing an implementation of BMaude in Maude and supporting different forms of behavioral deduction and reasoning have also been discussed.

Much work remains ahead. In particular, as pointed out by Diaconescu and Futatsugi [31]:

The practical significance of full HOSRWL (hidden order-sorted rewriting logic) is still little understood.

Indeed, we view gaining a fuller understanding of the behavioral aspects of rewriting logic as one of our key research priorities. More generally, we would like to advance our theoretical understanding, as well as the BMaude language design, its methods, and its applications, along the following lines:

• As already mentioned, the behavioral aspects of rewriting logic need to be better understood. This should incorporate the contributions already made by Diaconescu [26], probably relating them to ideas of Reichel [83] and Pattinson [79], and to recent advances in unifying the algebraic and coalgebraic approaches in relation to SOS and to concurrency theory [24].

• A better understanding of behavioral rewriting logic should also help in understanding how the hidden-sorted notion of object [50,10], as well as coalgebraic notions of object [83,62], can be unified with the distributed object concepts supported by rewriting logic [68].

• The integration of the different heterogeneous logics involved will probably require a more general treatment of heterogeneity than just the sublogic inclusion MEL $\hookrightarrow$ BMEL and the associated inclusions between categories.
of theories used in Section 5. The work of Diaconescu on Grothendieck institutions \[27\], as well as Mossakowski’s recent work on heterogeneous specifications \[74\] will certainly be very relevant. The relationships of our approach to the ideas of Reichel on how to allow arbitrary nestings of initiality and finality constraints, and his general treatment of these constraints by means of sketches \[84\], should also be investigated in detail.

• Another important area of future research is the integration of inference systems, formal tools, and formal methods. At present, the Maude tools support: (1) inductive theorem proving \[22\] and reflective versions of induction \[1\]; (2) Church-Rosser, coherence, and termination checking, and Knuth-Bendix completion \[22, 35, 33\]; (3) real-time specification, analysis and model checking \[76, 75\]; and (4) model checking of linear temporal logic properties satisfied by rewrite theories (already supported in the latest alpha versions of Maude 2.0). The question now is how all these inference systems, tools, and methods, should complement and work in conjunction with behavioral deduction methods and tools such as those described in Section 7 and others to be developed. We believe that a well-engineered and rigorous integration of such tools and methods—and of other techniques such as temporal logic deduction and abstraction—should make significantly easier the verification and formal analysis of distributed systems and should also support lighter methods such as specification-based testing, runtime verification, and monitoring.

• Last, but not least, the theory, the specification language design, and the formal tools and methods should be developed and tested in conjunction with applications and case studies; for example in areas such as communication protocols, security, distributed systems, and embedded systems.

References


