Journal of Universal Computer Science, vol. 6, no. 1 (2000), 74-96
submitted: 15/6/99, accepted: 2/11/99, appeared: 28/1/00 © Springer Pub. Co.

Behavioural Coherence in Object-Oriented Algebraic Specification[1, 2]

Răzvan Diaconescu
(Institute of Mathematics of the Romanian Academy
Email: diacon@stoilow.imar.ro.)

Kokichi Futatsugi
(Japan Advanced Institute of Science and Technology
Email: kokichi@jaist.ac.jp.)

Abstract: We extend the classical hidden algebra formalism by a re-arrangement of the basic concepts. This re-arrangement of the hidden algebra formalism permits an extension to novel concepts which bring new practical strength to the specification and verification methodologies. The main novel concept, which constitutes the core of this work, is that of *behavioural coherence*, which is essentially a property of preservation of behavioural structures. We define this concept and study its main denotational and computational properties, and also show how the extension of hidden algebra with behavioural coherence still accommodates the coinduction proof method advocated by classical hidden algebra and, very importantly, permits operations with several hidden sorts in the arity. The emphasis of this paper is however on the methodologies related to behavioural coherence. We present the basic methodologies of behavioural coherence by means of examples actually run under the CafeOBJ system, including many proofs with the system exiled to appendices.

Keywords: behavioural equivalence, hidden algebra, behavioural coherence.
Category: F.3.2.

1 Introduction

This is a paper both about algebraic specification and verification foundations and methodologies. It belongs to a modern trend in algebraic specification, called *behavioural specification*. There are several formalisms for behavioural specification, our research lies within the so-called *hidden algebra* [10, 9] formalism. This style of specification is *object-oriented* as opposed to the *data-oriented* classical algebraic specification style. It has been argued that behavioural abstraction might be practically very effective for specifying and verifying large systems (in general software, but not only). The main reasons for this are the simplicity of the object-oriented style of specification (which is mainly due to the behavioural abstraction mechanism in which the strict equality between the states of objects plays a secondary rôle, the primary equality being *observational*[3]) contrasting to the somehow tedious and low-level techniques required by the data-oriented

[1] C. S. Calude and G. Ştefănescu (eds.). *Automata, Logic, and Computability. Special issue dedicated to Professor Sergiu Rudeanu Festschrift.*

[2] *First version of this paper appeared as Technical Report IS-RR-98-0017F, Japan Advanced Institute for Science and Technology, ISBN 0918-7553, June 1998.*

[3] Two states of an object are observationally equal (or equivalent) if and only if any application on any string of "methods" gets the same values for the "attributes".

approach. In fact, it can be said that behavioural specification realizes algebraic specification as a true *specification* paradigm.

This work builds on classical hidden algebra which was started several years ago by Joseph Goguen. We extend classical hidden algebra formalism by a re-arrangement of the basic concepts. This re-arrangement of the hidden algebra formalism permits an extension to novel concepts which bring new practical strength to the specification and verification methodologies. The main novel concept, which constitutes the core of this work, is that of *behavioural coherence*, which is essentially a property of preservation of behavioural structures. We define this concept and study its main denotational and computational[4] properties, and also show how the extension of hidden algebra with behavioural coherence still accommodates the coinduction proof method advocated by classical hidden algebra. The emphasis of this paper is however on the methodologies related to behavioural coherence. We present the basic methodologies of behavioural coherence by means of examples actually run under the CafeOBJ system, including many proofs with the system exiled to appendices.

Our extended hidden algebra formalism constitutes the semantic foundation for the behavioural specification paradigm as realized in the new object-oriented algebraic specification language CafeOBJ [6]. In fact, all concepts defined here are faithfully reflected by the CafeOBJ formal definition and were introduced in a rather concise form in [6] which contains the formal definition and semantics of the CafeOBJ. Therefore, this paper serves also as an introduction to advanced behavioural specification with CafeOBJ. Other important publications on behavioural methodologies for algebraic specification in CafeOBJ include a general survey [7] presenting both the basic methodologies and the object composition ones, and [13] which focuses on the object composition methodology. In this context let us also mention that our extended hidden algebra formalism is highly convergent to the so-called "observational logic" of Bidoit and Hennicker [12].

Finally, let us enumerate several practical methodological benefits of behavioural coherence. The use of behaviourally coherent "methods" and "attributes" for object specification may result in big simplifications at the verification stage, while keeping smooth computational characteristics. Behavioural coherent "methods" and "attributes" can be also used effectively in a denotational rôle. Another important application area is that of "hidden" constructors on the states of objects, this methodology being particularly effective for specifying object non-determinism in a simple and elegant way. It should be noted that this use of behavioural coherence puts hidden algebra beyond the power of other behavioural specification formalisms, such as the so-called "co-algebra" [14].

1.1 Basic Algebra Concepts, Notations, and Terminology

In this section we review the basic concepts, notations, and terminology, which constitute now the folklore of algebraic specification. Although the hidden algebra formalism accommodates well (and even gets more power from) the order-sorted approach (see [3]), for reasons of simplicity of presentation, we develop all formal definition and results in a many-sorted framework.[5]

[4] Including a special concept of term rewriting emerging from our approach.
[5] This will not prevent us to use sub-sorting in the examples.

Given a **sort set** S, an S-**indexed** (or **sorted**) **set** A is a family $\{A_s\}_{s\in S}$ of sets indexed by the elements of S. In this context, $a \in A$ means that $a \in A_s$ for some $s \in S$. Similarly, $A \subseteq B$ means that $A_s \subseteq B_s$ for each $s \in S$, and an S-**indexed** (or **sorted**) **function** $f\colon A \to B$ is a family $\{f_s\colon A_s \to B_s\}_{s\in S}$. Also, we let S^* denote the set of all finite sequences of elements from S, with $[]$ the empty sequence. Given an S-indexed set A and $w = s_1...s_n \in S^*$, we let $A_w = A_{s_1} \times \cdots \times A_{s_n}$; in particular, we let $A_{[]} = \{\star\}$, some one point set. Also, for an S-sorted function $f\colon A \to B$, we let $f_w\colon A_w \to B_w$ denote the function product mapping a tuple of elements $(a_1, \ldots, a_n)$ to the tuple $(f_{s_1}(a_1), \ldots, f_{s_n}(a_n))$.

A (n S-**sorted**) **signature** (S, Σ) is an $S^* \times S$-indexed set $\Sigma = \{\Sigma_{w,s} \mid w \in S^*,\ s \in S\}$; we often write just Σ instead of (S, Σ). Note that this definition permits *overloading*, in that the sets $\Sigma_{w,s}$ need *not* be disjoint. Call $\sigma \in \Sigma_{[],s}$ a **constant symbol** of sort s. A **signature morphism** Φ from a signature (S, Σ) to a signature (S', Σ') is a pair (f, g) consisting of a map $f\colon S \to S'$ of sorts and an $S^* \times S$-indexed family of maps $g_{w,s}\colon \Sigma_{w,s} \to \Sigma'_{f^*(w),f(s)}$ on operation symbols, where $f^*\colon S^* \to S'^*$ is the extension of f to strings[6]. We may write $\Phi(s)$ for $f(s)$, $\Phi(w)$ for $f^*(w)$, and $\Phi(\sigma)$ for $g_{w,s}(\sigma)$ when $\sigma \in \Sigma_{w,s}$.

A Σ-**algebra** A consists of an S-indexed set A and a function $A_\sigma\colon A_w \to A_s$ for each $\sigma \in \Sigma_{w,s}$; the set A_s is called the **carrier** of A of sort s. If $\sigma \in \Sigma_{[],s}$ then A_σ determines a point in A_s which may also be denoted A_σ. A Σ-**homomorphism** from one Σ-algebra A to another B is an S-indexed function $h\colon A \to B$ such that

$$h_s(A_\sigma(a)) = B_\sigma(h_w(a))$$

for each $\sigma \in \Sigma_{w,s}$ and $a \in A_w$. (When $n = 0$, this condition just says that $f(A_\sigma) = B_\sigma$.) A Σ-homomorphism $h\colon A \to B$ is a Σ-**isomorphism** iff each function $h_s\colon A_s \to B_s$ is bijective (i.e., one-to-one and onto, in an older terminology).

Given a many sorted signature Σ, an S-indexed set X will be called a set of **variable symbols** if the sets X_s are disjoint from each other and from all of the sets $\Sigma_{w,s}$. Given a set X of variable symbols, we let $\Sigma(X)$ denote the signature formed by adding the elements of X to Σ as new constants, and we let $T_\Sigma(X)$ denote $T_{\Sigma(X)}$ viewed as a Σ-algebra. It is called the Σ-**term algebra** or **free** Σ-**algebra** generated by X, and has the property that if $\theta\colon X \to A$ is an **valuation**, i.e., a (S-sorted) function to a Σ-algebra A, then there is a unique extension of θ to a Σ-homomorphism $\theta^*\colon T_\Sigma(X) \to A$. (Strictly speaking, the usual term algebra is not free unless the constant symbols in Σ are mutually disjoint; however, even if they are not disjoint, a closely related term algebra, with each constant annotated by its sort, is free.) Also, we let T_Σ denote the **initial** term Σ-algebra $T_\Sigma(\emptyset)$, noting that this means there is a unique Σ-homomorphism $T_\Sigma \to A$ for any Σ-algebra A. Call $t \in T_\Sigma$ a **ground** Σ-**term**. When the unique Σ-homomorphism $T_\Sigma \to A$ is surjective, we call A a **reachable** algebra. Thus, each element of a reachable algebra can be denoted by a ground term. The Σ-terms (modulo renaming of the variables) can be regarded as **derived operations** by defining the arity $ar(t)$ for terms t by the following procedure:

[6] This extension is defined by $f^*([]) = []$ and $f^*(ws) = f^*(w)f(s)$, for w in S^* and s in S.

- consider the set $var(t)$ of all variables occurring within t,
- transform $var(t)$ into a string by fixing an arbitrary order on this set, and
- finally, replace the variables in the string previously obtained by their sorts.

If the arity of a term t is w, then for any Σ-algebra A we can define the interpretation of t as derived operation $A_t\colon A_w \to A_s$ by $A_t(a) = \theta^*(t)$ where $\theta\colon var(t) \to A$ is the valuation corresponding to the string $a \in A_w$.

A **Σ-context** $c[z]$ is a Σ-term c with a marked variable z occuring only once in c.

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

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

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

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

iff for all valuations $\theta\colon X \to A$, we have $\theta^*(t) = \theta^*(t')$ whenever $\theta^*(t_j) = \theta^*(t'_j)$ for $j = 1, ..., m$. Given a set E of (possibly conditional) Σ-equations, we call any Σ-algebra that satisfies E a (Σ, E)-algebra.

A **Σ-congruence** on a Σ-algebra A is an S-sorted family of relations, $\equiv_s$ on A_s, each of which is an equivalence relation, and which also satisfy the **congruence property**, that given any $\sigma \in \Sigma_{w,s}$ and any $a \in A_w$, then $A_\sigma(a) \equiv_s A_\sigma(a')$ whenever $a \equiv_w a'$.[7] The **quotient** of A by $\equiv$, denoted $A/\equiv$, has carriers $(A/\equiv)_s = A_s/\equiv_s$, which inherit a Σ-algebra structure by defining $A/\equiv_\sigma([a_1], \ldots, [a_n]) = [A_\sigma(a)]$, where $\sigma \in \Sigma_{w,s}$ and $a \in A_w$, where $[a]$ denotes the $\equiv$-equivalence class of a.

We now consider the *logic* of many sorted algebra, that is, rules for deducing new equations from old ones. Given a set E of (possibly conditional) Σ-equations, we define the (unconditional) Σ-equations that are **derivable** from E recursively, by the following **rules of deduction**:

(1) <u>Reflexivity</u>: Each equation $(\forall X)\, t = t$ is derivable.
(2) <u>Symmetry</u>: If $(\forall X)\, t = t'$ is derivable, then so is $(\forall X)\, t' = t$.
(3) <u>Transitivity</u>: If $(\forall X)\, t = t'$ and $(\forall X)\, t' = t''$ are derivable, then so is $(\forall X)\, t = t''$.
(4) <u>Congruence</u>: If $(\forall X)\, t_i = t'_i$ is derivable, where $t_i, t'_i \in T_\Sigma(X)_{s_i}$ for $i = 1, ..., n$, then for any $\sigma \in \Sigma_{s_1 \ldots s_n, s}$, the equation $(\forall X)\, \sigma(t_1, \ldots, t_n) = \sigma(t'_1, \ldots, t'_n)$ is also derivable.
(5) <u>Substitutivity</u>: Given $(\forall Y)\, t = t'$ **if** $t_1 = t'_1, \ldots, t_m = t'_m$ in E and given a substitution $\theta\colon Y \to T_\Sigma(X)$ such that $(\forall X)\, \theta^*(t_j) = \theta^*(t'_j)$ is derivable for $j = 1, ..., m$, then $(\forall X)\, \theta^*(t) = \theta^*(t')$ is also derivable.

[7] Meaning $a_i \equiv_{s_i} a'_i$ for $i = 1, ..., n$, where $w = s_1 \ldots s_n$ and $a = (a_1, \ldots, a_n)$.

Given a set E of Σ-equations, let $E^{\bigcirc}$ denote the S-sorted set of pairs (t,t') of *ground* Σ-terms such that $(\forall\emptyset)\, t = t'$ is derivable from E. Then $E^{\bigcirc}$ is a Σ-congruence by rules (1)–(4). The following completeness result was first proved by Goguen and Meseguer [11], although the unconditional one sorted form is very well known, going back to Birkhoff [2] in 1935:

Theorem 1. *Given a set E of (possibly conditional) Σ-equations, an unconditional Σ-equation is satisfied by every (Σ,E)-algebra iff it is derivable from E using the rules (1)–(5).*

Goguen and Meseguer [11] use the above to prove the following basic result:

Theorem 2. *The Σ-algebra $T_{\Sigma,E} = T_{\Sigma}/E^{\bigcirc}$ is an initial (Σ,E)-algebra, in the sense that for any (Σ,E)-algebra A there is a unique Σ-homomorphism $h\colon T_{\Sigma,E} \to A$.*

Of course, there are many other initial (Σ,E)-algebras, but they are all Σ-isomorphic to this one.

Now we briefly recall some basic rewriting concepts and notations. For simplicity we restrict the discussion to the unconditional case. Given a signature Σ, a Σ-**rule** is given by

$$(\forall)\, t \ \texttt{->}\ t'$$

where t,t' are Σ-terms such that $var(t') \subseteq var(t)$. A Σ-**TRS**[8] is a finite collection of Σ-rules. Given a fixed TRS, then a Σ-term t_0 **rewrites** (in one step) to the Σ-term t_1 iff there is a TRS-rule $(\forall)\, t \ \texttt{->}\ t'$ such that $t_0 = c[\theta^*(t)]$ and $t_1 = c[\theta^*(t')]$ for some Σ-context (called **rewrite context**) c and some substitution θ. This is denoted as $t_0 \to t_1$. The transitive-reflexive closure of $\to$ is denoted as $\stackrel{*}{\to}$. A TRS is **(ground) confluent** iff for any (ground) term t_0, if $t_0 \stackrel{*}{\to} t_1$ and $t_0 \stackrel{*}{\to} t_2$, then there exists t_3 such that $t_1 \stackrel{*}{\to} t_3$ and $t_2 \stackrel{*}{\to} t_3$, and it is **(ground) terminating** iff there are only finite rewrite chains from any (ground) term t. A term t is in **normal form** iff there is no rewrite from t. When the TRS is confluent and terminating, then each term t has a unique normal form $nf(t)$ such that $t \stackrel{*}{\to} nf(t)$.

Given a signature morphism $\Phi\colon \Sigma \to \Sigma'$ and a Σ'-algebra A', we can define the **reduct** of A' to Σ, denoted $\Phi(A')$ or $A'|_{\Phi}$, to have carriers $A'_{\Phi(s)}$ for $s \in S$, and to have operations $\Phi(A')_{\sigma}$ for $\sigma \in \Sigma_{w,s}$ defined by $\Phi(A')_{\sigma}(m) = A'_{\Phi(\sigma)}(m)$ for $m \in A'_{\Phi(w)}$. Also, given a Σ'-homomorphism $h\colon A'_1 \to A'_2$, we can define $h|_{\Phi}\colon A'_1|_{\Phi} \to A'_2|_{\Phi}$ by $(h|_{\Phi})_s = h_{\Phi(s)}$ for $s \in S$.

Similarly, given a Σ-equation e of the form $(\forall X)\, t = t'$, we define $\Phi(e)$ to be the Σ'-equation $(\forall X')\, \overline{\Phi}(t) = \overline{\Phi}(t')$, where X' is the S'-indexed set, also denoted $\Phi(X)$, with $X'_{s'} = \bigcup_{\Phi(s)=s'} X_s$ for $s' \in S'$, and where $\overline{\Phi}\colon T_{\Sigma(X)} \to T_{\Sigma'(X')}$ is the S-indexed function defined by viewing $T_{\Sigma'(X')}$ as a $\Sigma(X)$-algebra using the reduct construction given above, and then the initiality of $T_{\Sigma(X)}$.

An important property of these translations on algebras and equations under signature morphisms is called the **Satisfaction Condition**, which expresses the invariance of satisfaction under change of notation:

[8] Abbreviation for **term rewriting system**.

Theorem 3. *Given an signature morphism $\Phi\colon\ \Sigma \to \Sigma'$, a Σ'-algebra A', and a Σ-equation e, then*

$$\Phi(A') \models_{\Sigma} e \text{ iff } A' \models_{\Sigma'} \Phi(e).$$

This theorem was first proved in the original version of [8], and constitutes the basic axiom of the so-called **institutions**, which have recently been emerging as the mathematical structure underlying the modern level of algebraic specification theory.

2 Coherent Hidden Algebra

In this section we extend the hidden algebra formalism for behavioural specification (as defined in seminal papers such as [9, 10] and abbreviated here as **HA**). Our extension yields a suitable framework for the development of the novel concept of behavioural coherence, and also corresponds exactly to the semantics of the behavioural specification paradigm as realized in the algebraic specification language CafeOBJ [6]. We refer to this extension as *coherent hidden algebra* (abbreviated **CHA**).

Hidden algebra (formerly called "hidden sorted algebra") was invented by Goguen as an extension of the (order sorted) equational logic formalism underlying the modern theory of abstract data types and, generally, constituting the logical foundation for classical algebraic specification. One of the early papers which present hidden algebra is [9], while the basic reference for this area might now be the survey [10]. An institution-independent approach to HA can be found in [3]. Hidden algebra extends ordinary algebra with sorts representing *states* of objects rather that data elements and also introduces a new concept of satisfaction between models (algebras) and sentences, called *behavioural satisfaction*.

CHA extends HA by introducing explicit concepts of *behavioural operation* and *behavioural sentence*. Because of behavioural sentences, CHA does not need a special notation for behavioural satisfaction, that would be treated just as the satisfaction of behavioural sentences. This has the advantage of a unitary institution for CHA with a unique satisfaction relation for both strict and behavioural sentences and of allowing strict equations on hidden sorts.

Definition 4. A **CHA signature** is a tuple (H, V, Σ, Σ^b), where

- H and V are disjoint sets of **hidden** sorts and **visible** sorts, respectively,
- Σ is an $(H \cup V)$-sorted signature,
- $\Sigma^b \subseteq \Sigma$ is a subset of **behavioural operations** such that each $\sigma \in \Sigma^b_{w,s}$ has *exactly* one hidden sort in w.

Notice that very often we will shorten the notation (H, V, Σ, Σ^b) to (H, V, Σ), or just Σ, when no confusion is possible.

From a methodological perspective, the operations in Σ^b have object-oriented meaning, $\sigma \in \Sigma^b_{w,s}$ is thought as an **action** (or "method" in a more classical jargon) on the space (type) of states if s is hidden, and thought as **observation** (or "attribute" in a more classical jargon) if s is visible. The last condition says that the actions and observations act on (states of) single objects.

Definition 5. A **CHA signature morphism** $\Phi\colon (H, V, \Sigma, \Sigma^b) \to (H', V', \Sigma', \Sigma'^b)$ is a signature morphism $\Sigma \to \Sigma'$ such that

(M1) $\Phi(V) \subseteq V'$ and $\Phi(H) \subseteq H'$,
(M2) $\Phi(\Sigma^b) = \Sigma'^b$ and $\Phi^{-1}(\Sigma'^b) \subseteq \Sigma^b$,

These conditions say that hidden sorted signature morphisms preserve visibility and invisibility for both sorts and operations, and the $\Sigma'^b \subseteq \Phi(\Sigma^b)$ inclusion expresses the encapsulation of classes (in the sense that no new actions (methods) or observations (attributes) can be defined on an imported class)[9]. However, this last inclusion condition applies only to the case when signature morphisms are used as module imports (the so-called *horizontal* signature morphisms); when they model specification refinement this condition might be dropped (this case is called *vertical* signature morphism).

Now, we turn our attention to models.

Definition 6. Given a CHA signature Σ, the class of its models consists of all Σ-algebras A.

Definition 7. Given a CHA signature Σ, a **behavioural context**[10] is any Σ-context $c[z]$ such that all operations above z in c are behavioural.

While ordinary satisfaction corresponds to reasoning about strict equality, behavioural satisfaction corresponds to reasoning about *behavioural equivalence*, which can be regarded as a looser form of equality. Behavioural equivalence is the main concept underlying the behavioural abstraction mechanism of specifications based on hidden algebra, hence it plays a central rôle in CHA.

Definition 8. Given a Σ-algebra A, two elements (of the same sort s) a and a' are called **behaviourally equivalent**, denoted $a \sim_s a'$ (or just $a \sim a'$) iff

$$A_c(a) = A_c(a')^{11}$$

for all *visible* behavioural contexts c.

Remark that the behavioural equivalence is a $(H \cup V)$-sorted equivalence relation, and on the visible sorts the behavioural equivalence coincides with the (strict) equality relation.

The concept of behavioural equivalence leads also to a different notion of equation and satisfaction:

Definition 9. Given a CHA signature Σ, a **behavioural Σ-equation** is a sentence of the form

$$(\forall X)\, t \;\sim\; t' \;\;\textbf{if}\;\; t_1 \approx_1 t'_1, \ldots, t_m \approx_m t'_m$$

where each $\approx_i$ is either $=$ or $\sim$ for all $i \in \{1, \ldots, m\}$, all other symbols having the same meaning as for ordinary equations.

---

[9] Without it the Satisfaction Condition fails, for more details on the logical and computational relevance of this condition see [9].

[10] Notice that the CafeOBJ concept of "behavioural context" corresponds to "visible behavioural context" in the sense of this paper.

[11] Notice that this equality means an equality between functions $A_{w_1 w_2} \to A_{s'}$, where $c\colon w_1 s w_2 \to s'$ with $w_1, w_2 \in (H \cup V)^*$ and $s' \in V$.

Now, we are ready to define the notion of satisfaction for hidden algebra.

Definition 10. Given a CHA signature Σ and a Σ-algebra A, a behavioural equation $(\forall X)\ t\ \sim\ t'$ **if** $t_1 \approx_1 t'_1, \ldots, t_m \approx_m t'_m$ is **satisfied** (also denoted by $\models$) by A iff

$$\theta^*(t) \sim \theta^*(t') \text{ whenever } \theta^*(t_i) \approx_i \theta^*(t'_i) \text{ for all } i \in \{1, \ldots, m\}$$

for all valuations $\theta\colon X \to A$.

Note that the CHA satisfaction generalizes the HA behavioural satisfaction [10, 9] since the satisfaction of $(\forall X)\ t\ \sim\ t'$ **if** $t_1 \sim t'_1, \ldots, t_m \sim t'_m$ corresponds exactly to the HA *behavioural* satisfaction of
$(\forall X)\ t\ =\ t'$ **if** $t_1 = t'_1, \ldots, t_m = t'_m$.

Definition 11. Given a CHA signature Σ and a Σ-algebra A, a **hidden congruence** is an equivalence relation $\equiv$ on A which is identity on visible sorts and is a Σ^b-congruence.

The following result constitutes the foundations for the **coinduction** [10] proof method for hidden algebra. We do it here again, since the CHA formalism is an extension of HA.

Theorem 12. *Given a CHA signature Σ and a Σ-algebra A, the behavioural equivalence relation on A is the largest hidden congruence on A.*

Proof. Consider a hidden congruence $\equiv$ on A. We have to prove that for any elements a, a' of the same sort h, $a \equiv_h a'$ implies $a \sim_h a'$, i.e., $A_c(a) = A_c(a')$ for all visible behavioural contexts c. We prove this by induction on the length of the context c. We may assume that h is hidden, otherwise the conclusion follows directly from the definition of hidden congruences.

If the length of $c[z]$ is 1, then $c[z]$ is just of the form $\sigma(t, z)$ where $\sigma\colon vh \to s$ (with $v \in V^*$ and $h \in H$) is a visible sorted behavioural operation and $t \in (T_\Sigma(X))_v$ is a v-tuple of terms. Because $\equiv$ is a Σ^b-congruence, we have that $A_c(x, a) = A_\sigma(A_t(x), a) \equiv A_\sigma(A_t(x), a') = A_c(x, a')$ for any valuation $x \in A_{ar(t)}$. Because $\equiv$ is a hidden congruence and the sort of σ is visible we have that $A_\sigma(A_t(x), a) = A_\sigma(A_t(x), a')$, thus $A_c(x, a) = A_c(x, a')$ for all $x \in A_{ar(t)}$ follows. Therefore $A_c(a) = A_c(a')$ as functions $A_{ar(t)} \to A_s$.

If the length of c is greater than 1, then there exists a visible behavioural context $c'\colon wh' \to s$ (with $h' \in H, s \in V$, and $w \in (H \cup V)^*$ of length smaller than the length of c and a behavioural operation $\sigma\colon vh \to h'$ (with $v \in V^*$ and $h \in H$), such that $c[z] = c'[\sigma(t, z)]$ for $t \in (T_\Sigma(X))_v$ a v-tuple of terms. Therefore, for all $x \in A_{ar(t)}$ and $y \in A_w$, $A_c(y, x, a) = A_{c'}(y, A_\sigma(A_t(x), a))$ and $A_c(y, x, a') = A_{c'}(y, A_\sigma(A_t(x), a'))$. Because $\equiv$ is a *hidden* congruence, $A_\sigma(A_t(x), a) \equiv_{h'} A_\sigma(A_t(x), a')$, and by the induction hypothesis we get $A_c(a) = A_c(a')$ as functions $A_{ar(t);w} \to A_s$.

3 Behaviourally Coherent Operations

3.1 The Definition

Definition 13. Given a CHA signature Σ and a hidden Σ-algebra A, an operation $\sigma \in \Sigma - \Sigma^b$ is **behaviourally coherent for** A iff it preserves the behavioural equivalence relation on A, i.e., if and only if

$$A_\sigma(a) \sim_s A_\sigma(a') \text{ if } a \sim_w a'$$

for all $a, a' \in A_w$, where $\sigma \in (\Sigma - \Sigma^b)_{w,s}$.

Notice that the operations having only visible sorts in the arity are trivially behaviourally coherent for any Σ-algebra A, so we will omit them from our arguments.

Corollary 14. *If all operations in $\Sigma - \Sigma^b$ are behaviourally coherent, then $\sim$ is a Σ-congruence.*

Corollary 15. *Given a CHA signature Σ and a Σ-algebra A for which all operations in $\Sigma - \Sigma^b$ are behaviourally coherent, there exists another Σ-algebra $\overline{A}$ (called the* **behavioural image of** A*) such that*

$$A \models (\forall X)\, t \ \sim\ t' \ \ \textit{iff} \ \ \overline{A} \models (\forall X)\, t \ = \ t'$$

for all behavioural Σ-equations $(\forall X)\, t \ \sim\ t'$.

Proof. By Corollary 14, $\sim$ is a Σ-congruence. Let $\overline{A}$ be the quotient algebra $A/_\sim$, and $[\_]\colon A \to \overline{A}$ be the corresponding quotient algebra morphism.

First assume $\overline{A} \models (\forall X)\, t \ = \ t'$. Let $\theta\colon X \to A$ be an arbitrary valuation. By hypothesis we have that $(\theta; [\_])^*(t) = (\theta; [\_])^*(t')$, which means $[\theta^*(t)] = [\theta^*(t')]$, therefore $\theta^*(t) \sim \theta^*(t')$. This concludes that $A \models (\forall X)\, t \ \sim\ t'$.

Conversely, let $A \models (\forall X)\, t \ \sim\ t'$ and consider and arbitrary valuation $\overline{\theta}\colon X \to \overline{A}$. There exists a valuation $\theta\colon X \to A$ such that $\theta; [\_] = \overline{\theta}$. We have that $\overline{\theta}^*(t) = (\theta; [\_])^*(t) = [\theta^*(t)] = [\theta^*(t')] = \overline{\theta}^*(t')$, which concludes the proof of $\overline{A} \models (\forall X)\, t \ = \ t'$.

This result has a special significance, since it can be generalized to any kind of sentences. In this way, given a concept of sentence (which need not be equational, it can be Horn clause, full first order, second order, etc.) we can define on top of it a concept of behavioural sentence with a corresponding notion of (behavioural) satisfaction. This idea has been fully exploited in [3] for developing an institution-independent theory of behavioural specification generalizing the concrete hidden algebra. We have all reasons to believe that this can be further developed in order to incorporate CHA.

Related to above, it might be interesting to briefly present the concept of behavioural image of an algebra from a categorical perspective.

Proposition 16. *Consider a hidden signature (H, V, Σ, Σ^b), and let Σ' be its sub-signature without the non-behavioural operations having at least one hidden sort in the arity. Given a Σ-algebra A, we may consider the unique Σ'-homomorphism h to the final Σ'-algebra in the sub-category of Σ'-algebras A' with $A'|_{\Sigma^v} = A|_{\Sigma^v}$ (which always exists [3, 10]), where Σ^v is the maximal visible sub-signature $\Sigma^v \subseteq \Sigma$. We factor h as $e; i$ where e is surjective and i is an inclusion. Then, denote the image (target) of e as A'. A' is an algebra which is the quotient under behavioural equivalence of the reduct $A|_{\Sigma'}$. If all operations from $\Sigma - \Sigma^b$ are behaviourally coherent for A, then the quotient A' can be uniquely expanded to a Σ-algebra $\overline{A}$ which is a quotient of A. Then $\overline{A}$ is the behavioural image of A.*

3.2 Sound Deduction

In this section we show that in the presence of behavioural coherence, equational deduction is sound for behavioural equivalence. This constitutes the basis for the execution of languages implementing CHA; the next section is devoted to the term rewriting-based operational semantics of CHA.

Theorem 17. *Given a set E of (possibly behavioural) equations for a hidden signature Σ, the class of (Σ, E)-algebras for which all operations in $\Sigma - \Sigma^b$ are behaviourally coherent is included in the class of (Σ, E)-algebras for which the ordinary equational deduction rules are sound for behavioural equations.*

Proof. First we will show that all equational deduction rules besides Congruence are sound anyway. For Base this is obvious, and the soundness of rules (1)–(3) follow immediately from the fact that behavioural equivalence is indeed an equivalence relation. We concentrate now on the Substitutivity rule.

Consider an algebra A, and a behavioural equation (since for ordinary equations the argument holds by the soundness of ordinary equational logic) $(\forall Y)\ t \sim t'$ **if** $t_1 \sim t_1', \ldots, t_m \sim t_m'$ in E. Assume that for some substitution $\theta\colon Y \to T_\Sigma(X)$, $(\forall X)\ \theta^*(t_j) \sim \theta^*(t_j')$ is true for $j \in \{1, \ldots, m\}$. We have to prove that $(\forall X)\ \theta^*(t) \sim \theta^*(t')$ is also true. Pick up an arbitrary valuation $\psi\colon X \to A$. Then for all $j \in \{1, \ldots, m\}$ we have that $\psi^*(\theta^*(t_j)) \sim \psi^*(\theta^*(t_j'))$ which means $(\theta; \psi^*)^*(t_j) \sim (\theta; \psi^*)^*(t_j')$ for all $j \in \{1, \ldots, m\}$. Because $A \models (\forall Y)\ t \sim t'$ **if** $t_1 \sim t_1', \ldots, t_m \sim t_m'$, it follows that $(\theta; \psi^*)^*(t) \sim (\theta; \psi^*)^*(t')$, thus $\psi^*(\theta^*(t)) \sim \psi^*(\theta^*(t'))$.

Now, we focus on Congruence. We show that Congruence is sound whenever all operations from $\Sigma - \Sigma^b$ are coherent. Assume $(\forall X)\ t_i \sim t_i$ is true in A for $i = 1, ..., n$. We have to prove that $(\forall X)\ \sigma(t_1, \ldots, t_n) \sim \sigma(t_1', \ldots, t_n')$ is also true in A. Consider a valuation $\psi\colon X \to A$. Then $\psi^*(t_i) \sim \psi^*(t_i')$ for $i = 1, ..., n$. Because σ is coherent for A, we have that $A_\sigma(\psi^*(t_1), \ldots, \psi^*(t_n)) \sim A_\sigma(\psi^*(t_1'), \ldots, \psi^*(t_n'))$. By the homomorphism property of ψ^*, we have that $\psi^*(\sigma(t_1, \ldots, t_n)) \sim \psi^*(\sigma(t_1', \ldots, t_n'))$.

Corollary 18. *Given a set E of (possibly behavioural) equations for a hidden signature Σ, the class of reachable (Σ, E)-algebras for which all operations in $\Sigma - \Sigma^b$ are behaviourally coherent is exactly the class of reachable (Σ, E)-algebras for which the ordinary equational deduction rules are sound for behavioural sentences.*

Proof. Consider a reachable (Σ, E)-algebra A. We will show that if Congruence is sound, then all operations in $\Sigma - \Sigma^b$ are behaviourally coherent. Consider such an operation $\sigma\colon w \to s$ and $a, a' \in A_w$. Each component of either a or a' can be denoted by a ground term, therefore let $a = A_t$ and $a' = A_{t'}$, with $t, t' \in (T_\Sigma)_w$. Then $A \models (\forall\emptyset)\ t \ \sim\ t'$ (notice that this is a finite conjunction of behavioural equalities indexed by w). Now, we have only to apply the hypothesis for Congruence for σ, i.e., $A \models (\forall\emptyset)\ \sigma(t) \ \sim\ \sigma(t')$, which means $A_\sigma(a) \sim A_\sigma(a')$.

This Corollary constitutes the foundations for computing with behaviourally coherent operations. The following section is devoted to this issue.

3.3 Behavioural Rewriting

The operational semantics of CHA requires a more sophisticated notion of rewriting which takes special care of the use of behavioural sentences during the rewriting process.

The following definition extends the concept of behavioural context with behaviourally coherent operations.

Definition 19. Given a CHA signature Σ and a Σ-algebra A, a **behaviourally coherent context for** A is any Σ-context $c[z]$ such that all operations above[12] the marked variable z are either behavioural or behaviourally coherent for A.

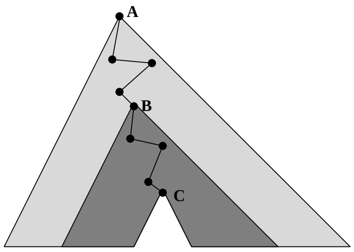
Notice that any behavioural context is also behaviourally coherent.

Proposition 20. *Consider a CHA signature Σ, a set E of Σ-sentences regarded as a TRS, and a Σ-algebra A satisfying the sentences in E. If t_0 is a ground term and for any rewrite step $t_0 \to t_1$ which uses a behavioural equation from E, the rewrite context has a visible behaviourally coherent sub-context for A, then $A \models (\forall\emptyset)\ t_0 \ = \ t_1$. If the rewrite context is behaviourally coherent for A, then $A \models (\forall\emptyset)\ t_0 \ \sim\ t_1$.*

Proof. We prove only the first case; the proof of the second case follows by a similar argument. There exists a behavioural equation $(\forall X)\ t \ \sim\ t'$ in E such that $t_0 = c[\theta^*(t)]$ and $t_1 = c[\theta^*(t')]$ for some rewrite context c and some valuation $\theta\colon X \to T_\Sigma$. Let c' be a visible behavioural coherent sub-context of c, this means $c[z] = c''[c'[z]]$ for some other rewrite context c''. Denote the unique Σ-homomorphism $T_\Sigma \to A$ by h. Therefore $A_{t_0} = A_{c''}(A_{c'}(h(\theta^*(t))))$ and $A_{t_1} = A_{c''}(A_{c'}(h(\theta^*(t'))))$. Because A satisfies the sentences in E, we also have that $h(\theta^*(t)) \sim h(\theta^*(t'))$. Because c' is a visible behaviourally coherent context for A, by induction on its length, we can prove that $A_{c'}(h(\theta^*(t))) = A_{c'}(h(\theta^*(t')))$. Then $A_{t_0} = A_{t_1}$, which means $A \models (\forall\emptyset)\ t_0 \ = \ t_1$.

This condition on rewriting was first introduced in [6], under the name of **behaviourally coherent context condition** and it is implemented by the CafeOBJ rewriting engine. It can be visualized by the following figure:

[12] Meaning that z is in the subterm determined by the operation.



Here A is the top position of the term to be reduced (represented by the big triangle), and C is the position of the sub-term (represented by the white triangle) to which the rule is applied. The rewriting context is represented by the whole gray area, and the behaviourally coherent sub-context by the dark gray area (with the top at B). The condition says that the sort of the operation at position B is visible, and that on the path between B and C there are no operations which are non-behavioural and not coherent.

4 Behavioural Coherence Methodologies

In this section we discuss several specification methodologies for behavioural coherence. We use the CafeOBJ notation for behavioural specification.

4.1 The Conservative Methodology

Consider the following parameterized specification of a buffer object with two methods (`take` and `put`) and two attributes (`get` and `empty?`).

We start by specifying the elements of the buffer:

```
mod! TRIV+(X :: TRIV) {
  op err :  -> ?Elt
}
```

The (initial denotation) module `TRIV+` is parameterized by the (loose denotation) built-in module `TRIV` which has only one sort `Elt`. The system also provides the built-in error super-sort `?Elt`. The denotation of `TRIV+` consists of all sets (as interpretation for `Elt`) plus an new element `err` of sort `?Elt` but outside the interpretation of `Elt`. In this specification, the sort `Elt` stands for the elements of the buffer, and `err` is an error value. `TRIV` becomes a parameter of (the below specification) `BUF1`, one can instantiate the elements of the buffer to any concrete set. Now, we can specify the buffer object:

```
mod* BUF1 {    protecting(TRIV+)
  *[ Buf ]*
  op init :  -> Buf
  op put : Elt Buf -> Buf   {coherent}
  bop get_ : Buf -> ?Elt
  bop take_ : Buf -> Buf
  op empty? : Buf -> Bool   {coherent}
  var E : Elt
  var B : Buf
  eq  empty?(init) = true .
```

```
  ceq empty?(take B) = true if empty?(B) .
  eq  empty?(put(E, B)) = false .
  ceq empty?(B) = true   if (get B) == err .
  bceq take put(E, B) = put(E, take B) if not empty?(B) .
  bceq take(put(E, B)) = B             if empty?(B) .
  ceq get B = err if empty?(B) .
  ceq get put(E, B) = E if empty?(B) .
  ceq get put(E, B) = get B if not empty?(B) .
}
```

The states of the buffer object are represented by the hidden sort `Buf` and there are only two behavioural operations (denoted by the keyword `bop`). The keywords `eq`, `ceq`, and `bceq` stand for (strict) unconditional equations, (strict) conditional equations, and conditional behavioural equations, respectively.

Notice that the predicate `empty?` (which checks the emptiness of the buffer) is specified as a Boolean-valued operation by using the built-in Boolean data type `BOOL` having one sort `Bool` with two constants (`true` and `false`) and the usual Boolean operations. The denotation of the `BOOL` data type consists of the initial algebra (more precisely, of the isomorphism class of initial algebras) interpreting the sort `Bool` as a set with only two elements, corresponding to the interpretations of the constants `true` and `false`.[13]

An interesting point of this specification is that one method (`put`) and one attribute (`empty?`) of the buffer object are declared as behaviourally coherent operations rather than behavioural operations. One important practical consequence of this is that the *definition*[14] of the behavioural equivalence relation gets drastically simplified, while the denotation of the specification remains unchanged. This is supported by the following proposition:

Proposition 21. *For each BUF1-model M, the operations* `put` *and* `empty?` *are behaviourally coherent.*

Proof. We first prove that `empty?` is behaviourally coherent. We have thus to show that $M_{\texttt{empty?}}(b) = M_{\texttt{empty?}}(b')$ whenever $b \sim_{\texttt{Buf}} b'$. From $b \sim_{\texttt{Buf}} b'$ we deduce that $M_{\texttt{get}}(b) = M_{\texttt{get}}(b')$. If both of them are $M_{\texttt{err}}$, then $M_{\texttt{empty?}}(b) = M_{\texttt{empty?}}(b') = M_{\texttt{true}}$ by the last equation on `empty?`. Otherwise, both of them are different than $M_{\texttt{err}}$, which means that $M_{\texttt{empty?}}(b) = M_{\texttt{empty?}}(b') = M_{\texttt{false}}$ by the first equation on `get` and because $M_{\texttt{Bool}}$ has only two elements, i.e., the interpretations of `true` and `false`.

Now, in order to complete the proof of this proposition we use the following lemma (its proof by using the CafeOBJ system is given in Appendix A.2):

Lemma 22. *The coherence of* `empty?` *implies the coherence of* `put`*.*

[13] The `BOOL` data type plays a crucial rôle for CafeOBJ conditional equations since their conditions are in fact `Bool`-sorted terms. This is more general than the classical definition of conditional equations (adopted also by this paper) and allows some forms of negation and dis-equality. Precisely speaking, the underlying logics of CafeOBJ are constrained over the built-in (or pre-defined) data type `BOOL`, and this is fully explained by the framework of *constraint logics* [5, 4].

[14] N.B. the behavioural equivalence relation remains the same, only its definition is simplified.

To resume, the operations `empty?` and `put` are behaviourally coherent in *all* `BUF1`-models, hence the denotation of `BUF1` is the same as the denotation when both `empty?` and `put` were specified as behavioural operations. This suggests the first (and in some sense the simplest) use of behavioural coherence:

> Some operations can be specified as behaviourally coherent rather than behavioural provided their coherence (with respect to the rest of the specification) can be proved. This results in a simplification of the definition of behavioural equivalence, with potential for simplifying the whole verification process related to this specification.

Notice that recently, Bidoit and Hennicker [1] gave practically relevant syntactic sufficient conditions for the conservative methodology.

4.2 The Non-Conservative Methodology

Now, we turn to a more sophisticated use of behavioural coherence. In the previous case, from the semantical perspective, the main point was that the coherence property held in all models. In other words, in that case the declaration "{`coherent`}" for operations it treated as a pure computational declaration[15] with no consequence on the denotations. In this section we explore a denotational rôle for such declarations.

The coherence declaration for an operation has the effect that the denotation of the specification is restricted to those models for which the corresponding operation is behaviourally coherent. Let us look again at the buffer example. The above `BUF1` specification leads to non-terminating computations due to the presence of the last equation on `empty?` and the first equation on `get`. But these equations are exactly the ones which ensure the coherence of `empty?`. The same situation can be achieved by dropping the last equation on `empty?` and by restricting the class of `BUF1`-models only to those for which `empty?` is coherent. Notice that by dropping the last equation on `empty?`, there are models for which `empty?` is *not* behaviourally coherent. Any model M with a buffer state b which for which $M_{\mathtt{get}}(b) = M_{\mathtt{err}}$ and with $M_{\mathtt{empty?}}(b) = \mathtt{false}$ would be such an example.[16]

So, consider the specification `BUF1` minus the last equation on `empty?`.

```
mod* BUF {    protecting(TRIV+)
  *[ Buf ]*
  op init :  -> Buf
  op put : Elt Buf -> Buf   {coherent}
  bop get_ : Buf -> ?Elt
  bop take_ : Buf -> Buf
  op empty? : Buf -> Bool   {coherent}
  var E : Elt
  var B : Buf
  eq  empty?(init) = true .
  ceq empty?(take B) = true if empty?(B) .
  eq  empty?(put(E, B)) = false .
  bceq take put(E, B) = put(E, take B) if not empty?(B) .
```

[15] Increasing the power of behavioural rewriting.

[16] Though such model cannot be a reachable one.

```
  bceq take(put(E, B)) = B             if empty?(B) .
  ceq get B = err if empty?(B) .
  ceq get put(E, B) = E if empty?(B) .
  ceq get put(E, B) = get B if not empty?(B) .
}
```

This specification avoids any non-termination, and its denotation is the same as that of BUF1. Lets denote by $\mathrm{MOD}(\mathtt{BUF})$ the denotation of BUF, by $\mathrm{MOD}(\mathtt{BUF1})$ the denotation of BUF1, and by $\mathrm{MOD}(\mathtt{BUF}')$ the denotation of the specification BUF' which is the same as BUF but without any coherence declarations for operations. We have that $\mathrm{MOD}(\mathtt{BUF}) \subset \mathrm{MOD}(\mathtt{BUF}')$ as strict inclusion. Also, $\mathrm{MOD}(\mathtt{BUF}) = \mathrm{MOD}(\mathtt{BUF1})$. The strictness of $\mathrm{MOD}(\mathtt{BUF}) \subset \mathrm{MOD}(\mathtt{BUF}')$ shows that the coherence declarations (in fact really only that of empty?) shrink the denotation to a smaller class of models, hence this is why this methodology is called "non-conservative".

The non-conservative methodology for behavioural coherence is strongly similar to the classical use of operation attributes (such as associativity (A), commutativity (C), identity (I), or idempotence (Z)) in ordinary algebraic specification. For example, imagine a specification of the data type of natural numbers with the plus operation declared commutative:

```
  op _+_ : Nat Nat -> Nat {comm}
```

In the case of the natural numbers such declaration is denotationally redundant since the commutativity of _+_ would be satisfied anyway by the standard (initial) model[17] which constitutes the denotation of the natural numbers data type. Hence, this use of "comm" declaration is a conservative methodology, the same as "coherent" declaration for put. However, the computational consequences of the "comm" declaration are crucial: by computing *modulo* commutativity the non-termination of computations is avoided. The same happens in the case of "coherent" declarations, the computation gets more power[18]. In the case of the coherence of empty? the similarity is almost perfect, since "coherent" declaration is used for avoiding non-terminating computations. On the side of non-conservative methodology, imagine a specification of monoids and a commutativity declaration for its binary operation:

```
  op _;_ : Mon Mon -> Mon {comm}
```

This declaration restricts the denotation only to the commutative monoids, thus having a similar denotational effect as the coherence declaration for empty?. In both cases the computational effect is maintained.

We may resume the non-conservative methodology by the following:

> Behavioural coherence declarations for operations restrict the denotation of the specification to the models for which these operations are behaviourally coherent, also giving more computational power. This usage of coherence declarations is similar to the usage of operation attributes (such as A,C,AC,I, etc.) in ordinary algebraic specification.

[17] This is a standard induction exercise in algebraic specification introductory texts.

[18] Due to the easier satisfaction of the so-called "behaviourally coherent context condition" check, see Appendix A.1.

As in the case of traditional A/C/I/Z attributes, coherence declarations should be used with some care because they might have an undesirable denotational impact. Abusing them might result in shrinking denotations too much, to the point of eliminating some desirable models (implementations). In general it is recommended to use the conservative methodology as much as possible, since this might simplify a lot the verification process without the burden to verify that certain implementations satisfy the coherence declarations (since in the conservative case these coherence properties are supposed to be proved at the abstract level of the specification). The non-conservative methodology is recommended for situations similar to the "`coherent`" declaration for `empty?`, when the shrink of the denotation is rather natural and helps with avoiding some computational problems.

4.3 The Hidden Constructor Methodology

The HA formalism requires that operations on hidden sorts have *at most* one hidden sort in their arity. This monadicity condition is essential for the case of behavioural operations (and in fact all hidden sorted operations in HA are meant as CHA behavioural operations) but may limit the specification power. Behaviourally coherent operations constitute the solution to this problem. Since they do not define the behavioural equivalence, they are not subject to the monadicity condition. On the other hand, they can be used effectively in behavioural specifications because they preserve the behavioural equivalence and thus they have smooth denotational and computational properties. We call such operations **hidden constructors**. Hidden cosntructors play a similar rôle in object-oriented algebraic specification as classical constructors play in ordinary (data type oriented) algebraic specification.

The hidden constructor methodology is "orthogonal" to the conservative vs. non-conservative methodologies in the sense that the behavioural coherence of a hidden constructor might be a consequence of the rest of the specification, or its coherence declaration might really shrink the denotation of the specification.

We illustrate this methodology by the specification of an *unreliable* buffer object. This means that there is a "put" method on the buffer object which is unreliable in the sense that the element which is put into the buffer might be lost.[19] We reuse the above "reliable" buffer object specification, the unreliable buffer object being thought as a refinement of the reliable buffer.

```
mod* UBUF {    protecting(BUF)
  *[ Buf < UBuf ]*
  op put  : Elt  UBuf -> UBuf  {coherent}
  bop take_ : UBuf -> UBuf
  op _|_  : UBuf UBuf -> UBuf  {coherent}
  op put? : Elt  UBuf -> UBuf  {coherent}
  op  get? :  Buf ?Elt -> Bool  {coherent}
  bop get? : UBuf ?Elt -> Bool
  var B : Buf
  vars U1 U2 U : UBuf
  var E  :  Elt
  var E' : ?Elt
```

[19] These kinds of "unreliable" objects are very useful for protocols specification and verification.

```
  eq put (E, U1 | U2)  =  put(E, U1) |  put(E, U2) .
  eq put?(E, U1 | U2)  = put?(E, U1) | put?(E, U2) .
  eq take(   U1 | U2)  = (take   U1) | (take   U2) .
  eq get?(   U1 | U2, E') = get?(U1, E') or get?(U2, E') .
  eq put?(E, U) = U | put(E, U) .
  eq get?(B, E') = (E' == get B) .
}
```

The states of the unreliable buffer object (represented by the sort `UBuf`) are thought as a non-deterministic extension of the states of the reliable buffer with the hidden constructor _|_ "building" the non-deterministic states of the unreliable buffer. The reliable buffer methods are extended to the unreliable buffer, an unreliable put method is introduced (`put?`), and, in the unreliable case, the "get" attribute becomes a relation (`get?`) rather than a function. Notice that the last equation expresses the fact that `get?` is an actual extension of `get` to the unreliable case, and the equation before the last one expresses the non-deterministic relationship between the unreliable "put" method and the reliable one. Notice also that `get?` is specified as behavioural operation since it is thought as an extension of a behavioural operation. The coherence of _|_, `put`, `put?`, and `get?` (on `Buf`) can be proved from the specification (we leave this as exercise to the reader). However, the coherence of the hidden constructor _|_ deserves special mention. This is a consequence of the four (strict) equations specifying the behaviour of _|_ with respect to the application of the "methods" and "attributes" of the unreliable buffer object. The strictness of these four equation is a matter of style rather than of methodology, for this specification we think that the implementations should strictly satisfy those equations.

The hidden constructor _|_ also has some useful properties, such as behavioural associativity, commutativity, and idempotence. The proofs of these can be seen in Appendix A.3. Also, in Appendix A.4 we present some proofs about the unreliable buffer object.

The hidden constructor methodology can be resumed as:

> Operations on hidden sorts having several hidden sorts in the arity might be effectively used in specifications provided they are behaviourally coherent. It is recommended to use them in conjunction with a conservative methodology, i.e., their coherence property is a consequence of the rest of the specification.

5 Conclusions and Future Work

We extended the traditional HA to a more powerful behavioural specification formalism (CHA) which includes explicit concepts of behavioural operation and behavioural sentence and also permits operations with several hidden sorts in the arity. We defined the novel concept of behaviourally coherent operation, studied its basic denotational and computational properties, and presented its basic methodologies via several CafeOBJ examples.

Further work will be dedicated for testing these methodologies for larger CASE studies. Work in this direction is already under development by the CafeOBJ team.

Acknowledgments

Many thanks to Joseph Goguen for constant encouragement of this research, for useful discussions of notation and terminology, and for his strong interest in coherence concepts. We wish also to thank the LDL group at JAIST for useful feedback and discussions on this research, and for providing an excellent research environment. We are very grateful to Toshimi Sawada, the implementor of the CafeOBJ system, for his implementation of behavioural specification paradigm in the CafeOBJ system, including the concepts introduced in this paper. This implementation supported crucially the research reported here.

References

[1] Michel Bidoit and Rolf Hennicker. Observer complete definitions are behaviourally coherent. In K. Futatsugi, J. Goguen, and J. Meseguer, editors, *OBJ/CafeOBJ/Maude at Formal Methods '99*, pages 83–94. Theta, Bucharest, 1999.

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

[3] Rod Burstall and Răzvan Diaconescu. Hiding and behaviour: an institutional approach. In A. William Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*, pages 75–92. Prentice-Hall, 1994. Also in Technical Report ECS-LFCS-8892-253, Laboratory for Foundations of Computer Science, University of Edinburgh, 1992.

[4] Răzvan Diaconescu. Category-based semantics for equational and constraint logic programming, 1994. DPhil thesis, University of Oxford.

[5] Răzvan Diaconescu. A category-based equational logic semantics to constraint programming. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *Recent Trends in Data Type Specification*, volume 1130 of *Lecture Notes in Computer Science*, pages 200–221. Springer, 1996. Proceedings of 11th Workshop on Specification of Abstract Data Types. Oslo, Norway, September 1995.

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

[7] Răzvan Diaconescu, Kokichi Futatsugi, and Shusaku Iida. Component-based algebraic specification and verification in CafeOBJ. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *FM'99 – Formal Methods*, volume 1709 of *Lecture Notes in Computer Science*, pages 1644–1663. Springer, 1999.

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

[9] Joseph Goguen and Răzvan Diaconescu. Towards an algebraic semantics for the object paradigm. In Harmut Ehrig and Fernando Orejas, editors, *Recent Trends in Data Type Specification*, volume 785 of *Lecture Notes in Computer Science*, pages 1–34. Springer, 1994.

[10] Joseph Goguen and Grant Malcolm. A hidden agenda. Technical Report CS97-538, University of California at San Diego, 1997.

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

[12] Rolf Hennicker and Michel Bidoit. Observational logic. In A. M. Haeberer, editor, *Algebraic Methodology and Software Technology*, number 1584 in LNCS, pages 263–277. Springer, 1999. Proc. AMAST'99.

[13] Shusaku Iida, Kokichi Futatsugi, and Răzvan Diaconescu. Component-based algebraic specification: - behavioural specification for component-based software engineering -. In *Behavioral specifications of businesses and systems*, pages 103–119. Kluwer, 1999.

[14] B. Jacobs and J.M. Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of EATCS*, 62:222–259, 1997.

A CafeOBJ Proofs

A.1 Using CafeOBJ for Behavioural Proofs

As mentioned above, CafeOBJ specifications can be executed. The basic execution mechanism of CafeOBJ is term rewriting by interpreting the specification equations as rewrite rules. For verifying behavioural properties in CafeOBJ, one writes a proof score accordingly to some proof method (such as coinduction), but the basic execution is done by rewriting. However, the CafeOBJ rewriting mechanism has some special features related to the special behavioural specification features of the language.

In CafeOBJ, there is a clear distinction between the strict equality (denoted syntactically by `eq` and supported semantically by the strict equality predicate `==`) and the behavioural equivalence (denoted syntactically by `beq` and supported semantically by the behavioural equivalence predicate `=b=`). The most basic execution command, called `reduce`, corresponds to reducing the input term to a normal form which is thought as *strictly* equal to the input term under the specification, thus it is conceptually linked to `==`. There is also a behavioural counterpart to this command, called `beh-reduce`, conceptually linked to `=b=`, but this is much less used in practice.[20]

When executing by `reduce` or when evaluating the predicate `==` special care should be taken with respect to the use of behavioural equations as rewrite rules. The reason is that they denote behavioural rather than strict equality, thus their application might dilute the strict equality into a behavioural equality. The so-called behaviourally coherence context condition mentioned in Section 3.3 ensures the safety of use of behavioural equalities as rewrite rules.

A.2 Proof of Lemma 22

In this appendix section we prove that the coherence of the `empty?` attribute of the buffer specification implies the coherence of the `put` method. We do this proof by the CafeOBJ system.

We first encode the behavioural equivalence relation in CafeOBJ:

```
mod! BARE-NAT {
  [ NzNat Zero < Nat ]
  op 0 : -> Zero
```

[20] There are several reasons for this. One of them is that the current proof methods for behavioural equivalence rarely require the direct use of `beh-reduce` or `=b=`. Another reason lies in the inherent incompleteness of the evaluation of `=b=`.

```
  op s_ : Nat -> NzNat
}
mod* BUF-BEQ {   protecting(BUF + BARE-NAT)
  op _R[_]_ : Buf Nat Buf -> Bool {coherent}
  bop take : Nat Buf -> Buf
  var N : Nat
  vars B B1 B2 : Buf
  eq take(0, B) = B .
  eq take(s(N), B) = take(N, take B) .
  eq B1 R[N] B2 = get take(N, B1) == get take(N, B2) .
}
```

The module BARE-NAT specifies a very simple data type for the natural numbers. The operation take introduced in BUF-BEQ is a second order generalization of the method take of BUF, which is necessary for defining the behavioural equivalence. The behavioural equivalence on Buf is defined by the parameterized relation _R[_]_.

The following is the proof score for Lemma 22:

```
open BUF-BEQ .
  ops b1 b1' b2 b2' : -> Buf .
  vars B1 B2 : Buf
  op n : -> Nat .
  op e : -> Elt .
```

The following are the assumptions corresponding to the case when both buffers are empty or to the case when both buffers are non-empty. These are the only cases to be considered because of the coherence of empty?.

```
  beq b1  = b2  .
  beq b1' = b2' .
  eq empty?(b2)  = true .
  eq empty?(b2') = false .
```

In the first case the proof of the coherence of put is given by the following reductions (notice also the case analysis corresponding to the parameter of _R[_]_):

```
red put(e, b1) R[0]   put(e, b2) .
red put(e, b1) R[s n] put(e, b2) .
```

For the second case, we need to proceed by induction. Here is the base case:

```
red put(e, b1') R[0]   put(e, b2') .
```

For the inductive step, we first assume the induction hypothesis

```
  cq get(take(n, put(e, B1))) == get(take(n, put(e, B2))) = true
     if B1 =b= B2 .
```

and we then perform the following reduction:

```
red put(e, b1') R[s n] put(e, b2') .
```

A.3 Proofs of behavioural ACZ properties of _|_

In order to prove the ACZ of _|_ as *behavioural* properties, we extend the CafeOBJ encoding of behavioural equivalence from the reliable buffer object to the unreliable one.

```
mod* UBUF-BEQ {   protecting(UBUF + BUF-BEQ)
  op _R[_,_]_ : UBuf Nat ?Elt UBuf -> Bool {coherent}
  bop take : Nat UBuf -> UBuf
  var N : Nat
  var E : ?Elt
  vars U U1 U2 : UBuf
  eq take(0, U) = U .
  eq [take] : take(s(N), U) = take(N, take U) .
  eq  U1 R[N,E] U2 = get?(take(N, U1), E) == get?(take(N, U2), E) .
}
```

We then build an environment for proofs:

```
mod* UBUF-PROOF {   protecting(UBUF-BEQ)
  ops u u' u'' : -> UBuf
  op n  : -> Nat
  op e  : ->  Elt
  op e' : -> ?Elt
  vars U1 U2 U : UBuf
  var E : Elt
  var N : Nat
}
```

We need to prove a lemma:

Lemma 23. *take(N, U1 | U2) = take(N, U1) | take(N, U2)*

Proof. We prove this by induction on the natural number parameter:

```
open UBUF-PROOF .
```

This is the base case:

```
red take(0, u | u') == take(0, u) | take(0, u') .
```

Now we assume the induction hypothesis:

```
  eq take(n, U1 | U2) = take(n, U1) | take(n, U2) .
```

and then do the induction step:

```
red take(s(n), u | u') == take(s(n), u) | take(s(n), u') .
close
```

Now we can proceed with the main proof.

```
open UBUF-PROOF .
```

by using Lemma 23:

```
  eq take(N, U1 | U2) = take(N, U1) | take(N, U2) .
```

and then prove the ACZ behavioural properties:

```
red (u | u')        R[n,e']  (u' | u) .
red (u | u') | u''  R[n,e']  u | (u' | u'') .
red u | u           R[n,e']  u .
close
```

A.4 Proofs about the Unreliable Buffer

In this appendix section we prove some (behavioural) properties of the unreliable buffer. Firstly, we do some testing reduction in order to get a feeling about how the unreliable buffer works. We just show the output from the CafeOBJ interpreter:

```
-- opening module UBUF(X.TRIV+).. done._*
-- reduce  put?(e1,put?(e2,init))
(init | put(e1,init)) | (put(e2,init) | put(e1,put(e2,init))) : UBuf
-- reduce  get?(put?(e1,put?(e2,init)),e1)
true : Bool
-- reduce  get?(put?(e1, put?(e2, init)), e2)
true : Bool
-- reduce  get?(put?(e1, put?(e2, init)), err)
true : Bool
-- reduce get?(put?(e1, put?(e2, b)),err)
false : Bool
-- reduce get?(take put?(e1, put?(e2, init)), e1)
true : Bool
-- reduce get?(take put?(e1, put?(e2, init)), e2)
false : Bool
-- reduce take put(e1, put?(e2, init)) == take put?(e1, put(e2, init))
false : Bool
-- reduce take put(e1, put?(e2, init)) =b= take put?(e1, put(e2, init))
true : Bool
```

Now we prove a true concurrency property between the reliable and unreliable "put" methods. This can be formulated as

$$\mathtt{put}(\mathtt{e},\mathtt{put?}(\mathtt{e},\mathtt{u})) \sim \mathtt{put?}(\mathtt{e},\mathtt{put}(\mathtt{e},\mathtt{u}))$$

for each unreliable buffer state u and each element e. Here is the proof:

```
open UBUF-PROOF .
```

We assume a previously proved lemma:

```
  eq take(N, U1 | U2) = take(N, U1) | take(N, U2) .
```

and then perform the corresponding reductions by taking care of a small case analysis:

```
red put(e, put?(e, u)) R[n,e]  put?(e, put(e, u)) .
red put(e, put?(e, u)) R[n,e'] put?(e, put(e, u)) .
close
```

The reachable unreliable buffer objects are richer in properties. For example the following constitute a complete axiomatization of the attribute `get?`. Notice that in the unreliable case the same is not possible for `take`.

```
mod* UBUF! {   protecting(UBUF)
  vars E E' : Elt
  var U : UBuf
  eq get?(put(E, U), err) = false .
  cq get?(put(E, U), E') = get?(U, E')
     if not(get?(U, err)) or (E =/= E' and get?(U, err)) .
  cq get?(put(E, U), E) = true   if get?(U, err) .
  cq get?(take U, err) = true  if get?(U, err) .
  bceq take put(E, U) = put(E, take U)  if not get?(U, err) .
}
```

We leave the proof of these properties for the reachable case to the reader.

Now we can concentrate to prove a last property for reachable unreliable buffer object models:

$$\texttt{take put?}(\texttt{e},\texttt{u}) \sim \texttt{put?}(\texttt{e},\texttt{take u}) \text{ if } \texttt{get?}(\texttt{u},\texttt{err}) \text{ is false}$$

We open the environment for the reachable unreliable buffer object and assume the hypothesis:

```
open UBUF-PROOF + UBUF! .
  eq get?(u, err) = false .
```

and then perform the reductions:

```
red take put?(e, u) R[n,e]  put?(e, take u) .
red take put?(e, u) R[n,e'] put?(e, take u) .
close
```