Specifying Coalgebras with Modal Logic

Alexander Kurz

Institut für Informatik, Ludwig-Maximilians-Universität München, Oettingenstr. 67, D-80538 München, Germany

Abstract

We propose to use modal logic as a logic for coalgebras and discuss it in view of the work done on coalgebras as a semantics of object-oriented programming. Two approaches are taken: First, standard concepts of modal logic are applied to coalgebras. For a certain kind of functor it is shown that the logic exactly captures the notion of bisimulation and a complete calculus is given. Examples of verifications of object properties are given. Second, we discuss the relationship of this approach with the coalgebraic logic of Moss [19].

1 Introduction

Coalgebras have been used in Reichel [20] and Jacobs [15] to formalise the notion of classes and objects in object-oriented programming. As for algebras, they use equational logic to specify coalgebras (i.e., classes and objects). An account of the connection between equational specifications and coalgebras is given by Hensel and Reichel [11] and Jacobs [14]. In this paper I propose a different approach: to use modal logic for the specification of coalgebras. The reasons are the following.

First, coalgebras are generalisations of transition systems and modal logic has been a natural choice whenever a logic for some transition systems was needed. It is therefore tempting to explore the relations between coalgebras and modal logic.

Second, coalgebras are used here to describe classes. Roughly speaking, given a coalgebra $(S, f : S \to FS)$ (S a set, F a functor, f a function) the state of an object is represented by an element $s \in S$. Now, looking for a logic to specify methods we should respect the idea of encapsulation: We do not want to talk about states, which are supposed to be non-observable, but only about observable behaviour. Modal logic is an obvious choice: Formulas of modal logic are evaluated in states but generally do not refer explicitly to specific states. Compared to equational logic a conceptual advantage is that equations between states can be avoided.

We will discuss two approaches to use the ideas of modal logic. First, given a certain kind of functor, find a translation of the corresponding coalgebras to Kripke models. Then apply results of modal logic to the Kripke models and transfer them back to coalgebras. This has the advantage that the well-developed machinery of modal logic can easily be used but the drawback that the translation does not generalise to arbitrary functors. Therefore, the second approach is to use the coalgebraic logic of Moss [19]. This logic has the advantage that its syntax is derived from the functor itself and does not depend on a non-canonical detour via Kripke models.

A third approach, due to Martin Rößiger [21], may be viewed as an intermediate one. By analysing functors as syntax trees, he manages to give a systematic description of modal logics for a larger class of functors than it is done in this paper. On the other side, as an advantage to Moss' approach, the logics still contain the intuitive modal operators. Comparing to our work, one should note that Rößiger's results could also be obtained via a translation, albeit a more complicated one, of coalgebras into Kripke models.

The paper is organised as follows. Sections 2 and 3 review the essentials on coalgebras and modal logic needed here. Section 4 introduces our modal language for coalgebras and relates modally definable classes of coalgebras to final coalgebras. Section 5 shows that the expressive power of the logic allows to define elements of coalgebras up to bisimulation and then a complete calculus for the logic is given. We also comment on the relation of the canonical model for a modal logic and the final coalgebra. Section 6 shows by examples that the logic allows for natural proofs of properties of programs. Section 7 relates our modal language to Moss' coalgebraic logic. Section 8 discusses the approach of this paper in view of object oriented programming.

2 Coalgebras

We recall the basic definitions, show how coalgebras may be used to describe objects and classes, and introduce the examples that are used in this paper. We mainly follow Jacobs [15].

Set is the category of sets with functions as morphisms. We only consider coalgebras in the category $\mathbf{Set}_F$, i.e., a coalgebra is a pair $(S, f : S \to FS)$ where S a set, F a functor on **Set** and f a function.[1] Coalgebras can be used to describe classes and objects. The functor F specifies the type of the methods, f the effect of the methods. The state of an object is represented by an element $s \in S$, $f(s)$ describing the results of the methods when sent to s.

[1] For the definition of a morphism in $\mathbf{Set}_F$ see the end of this section.

The functors we consider are of the form

$$F(S) = (B_1 + C_1 \times S)^{A_1} \times \dots (B_n + C_n \times S)^{A_n}.$$

How this describes the number and type of the methods of the class will become clear in the following example. Let us consider a one-cell buffer with two operations *store* and *read*. *store* is supposed to put an element in the buffer, *read* should output the current element or yield an error message if the buffer is empty. Writing A for the set of elements that the buffer may contain and 1 for the one-element set containing the error message, the signature becomes

$$store : S \times A \to S, read : S \to 1 + A \times S.$$

Using the isomorphism $(S \times A \to S) \simeq (S \to S^A)$ we can write *store* and *read* as one function

$$\langle store, read \rangle : S \to S^A \times (1 + A \times S).$$

That is, the functor F in our example is given by

$$F(S) = S^A \times (1 + A \times S).$$

The second example we will consider is a LIFO-queue with two operations $in : S \times A \to S$ and $out : S \to 1 + A \times S$ where again A denotes the set of possible elements to be stored in the queue and 1 is the set containing the error message. Both examples have the same functor. That their behaviour is different will be expressed by the modal logic presented in section 4.

One of the advantages of viewing transition systems as coalgebras is that, once the functor is given, there is a canonical notion of bisimulation. This is due to the fact that functors are not only defined on sets but also on functions: Let $F(S) = (B_1 + C_1 \times S)^{A_1} \times \dots (B_n + C_n \times S)^{A_n}$ and $h : S \to S'$ be a function, then $Fh : FS \to FS'$ takes a tuple of functions $(g_1, \dots g_n), g_i \in A_i \to B_i + C_i \times S$ to a tuple of functions $(g'_1, \dots g'_n), g'_i \in A_i \to B_i + C_i \times S'$, where the g'_i are defined as follows (a, b, c, t are in A_i, B_i, C_i, S, respectively):

$$g'_i(a) = \begin{cases} b & \text{if } g_i(a) = b \\ (c, h(t)) & \text{if } g_i(a) = (c, t) \end{cases}$$

Now, we can define a *homomorphism* (or *morphism* for short) between two coalgebras $(S, f), (S', f') \in \mathbf{Set}_F$ to be a function $h : S \to S'$ s.t. $f' \circ h = Fh \circ f$. It is not difficult to show that this definition of a morphism of coalgebras corresponds to the usual definition of a functional bisimulation. A *bisimulation* between two coalgebras $(S, f), (T, g)$ is a relation $R \subset S \times T$ such that there exists a function $r : R \to FR$ together with two coalgebra morphisms $(R, r) \to (S, f)$ and $(R, r) \to (T, g)$. R is a *total bisimulation* iff these two coalgebra morphisms are surjective. We write $(S, s) \simeq^F (T, t)$ iff there is a bisimulation between $(S, f), (T, g)$ containing (s, t).

To simplify notation we assume the sets B_i, C_i to be disjoint. Also, we will write S for (S, f) if there is no danger of confusion.

3 Modal Logic

In this section we review some basics of modal logic needed later.[2] The modal logics considered in this paper are extensions of propositional logic. A modal language $\mathcal{L}$ consists of the propositional connectives, a set of atomic propositions $\mathbf{P}$ and a set of unary modal operators that we write as $\{[i] : i \in I\}$, I a set.

Given a modal language with atomic propositions $\mathbf{P}$ and modal operators $\{[i] : i \in I\}$, a *Kripke frame* $(W, \mathcal{R})$ for this language is given by a set of worlds W (also called states) and a family of relations $\mathcal{R} = (R_i)_{i \in I}, R_i \subset W \times W$. wR_iv should be read as "there is a transition labelled i from state w to state v". A *Kripke model* $(W, \mathcal{R}, V)$ for the given language consists of a Kripke frame $(W, \mathcal{R})$ and a mapping, called valuation, $V : \mathbf{P} \to \mathcal{P}(W)$ that assigns to every atomic proposition a set of worlds. The natural notion of a morphism of Kripke models is that of a functional bisimulation, traditionally called *p-morphism*. A p-morphism $f : (W, \mathcal{R}, V) \to (W', \mathcal{R}', V')$ is a function $f : W \to W'$ satisfying (i) $wR_iv \;\Rightarrow\; f(w)R'_if(v)$, (ii) $f(w)R'_iv' \;\Rightarrow\; \exists v : wR_iv \;\&\; f(v) = v'$ and (iii) $w \in V(p) \;\Leftrightarrow\; f(w) \in V'(p)$.

The semantics of modal logic in terms of transition systems is defined as follows. Given a formula φ of the language and a Kripke model for the language $M = (W, \mathcal{R}, V)$ and a world $w \in W$ of the model, the relation $M, w \models \varphi$ is defined for the propositional connectives as to be expected and for atomic propositions and modal operators as follows:

$$\begin{array}{lll} M, w \models p & \text{iff} & p \in \mathbf{P} \text{ and } w \in V(p) \\ M, w \models [i]\varphi & \text{iff} & \forall v : wR_iv \;\Rightarrow\; M, v \models \varphi \end{array}$$

As usual, $M \models \varphi$ is defined by quantifying over all worlds, and $\models \varphi$ iff $M \models \varphi$ for all M. Similarly, given a frame $T = (W, \mathcal{R})$ and $w \in W$, $T, w \models \varphi$ iff $(W, \mathcal{R}, V), w \models \varphi$ for all valuations $V : \mathbf{P} \to \mathcal{P}(W)$. $T \models \varphi$ iff $T, w \models \varphi$ for all $w \in W$. For Γ a set of formulas and φ a formula, the *consequence relation* $\Gamma \models \varphi$ is to be understood in its global sense, that is, $\Gamma \models \varphi \;\Leftrightarrow\; \forall M(M \models \Gamma \;\Rightarrow\; M \models \varphi)$. The *theory* of a world w in a model (or frame) S is $\mathrm{Th}(S, w) = \{\varphi : S, w \models \varphi\}$ and $\mathrm{Th}(S) = \{\varphi : S \models \varphi\}$. Two models or frames

[2] For more details see e.g. Goldblatt [5].

are called *logically equivalent* (or sometimes *equivalent* for short) iff they have the same theory.

p-Morphisms preserve and reflect satisfaction of formulas in every world of a model. Moreover, if $f : M \to M'$ a (surjective) p-morphism then $M \models \varphi$ if (and only if) $M' \models \varphi$.

Given a modal language $\mathcal{L}$, the modal logic determined by all Kripke models for $\mathcal{L}$ is named $\mathbf{K}_{\mathcal{L}}$, i.e., $\mathbf{K}_{\mathcal{L}} = \{\varphi : M \models \varphi$ for all models M for $\mathcal{L}\}$. $\mathbf{K}_{\mathcal{L}}$ has a strongly complete axiomatisation given by the axioms and rules below. To simplify notation we use $\Box$ as syntactic variable for the modal operators $[i]$.

(taut)	all propositional tautologies
(dist)	$\Box(\varphi \to \psi) \to \Box\varphi \to \Box\psi$ for all $\Box$ of $\mathcal{L}$
(mp)	from $\varphi, \varphi \to \psi$ derive ψ
(nec)	from φ derive $\Box\varphi$ for all $\Box$ of $\mathcal{L}$

Strictly speaking, (dist) is not an axiom but an *axiom scheme*, i.e. all instances of (dist) with formulas of $\mathcal{L}$ substituted for φ, ψ are axioms. Following a common abuse of language we will often simply speak of axioms.

For Φ a set of formulas and φ a formula, $\Phi \vdash \varphi$ means that there is a finite derivation of φ using only the axioms and rules and the formulas in Φ. The above calculus is sound and strongly complete, that is, $\Phi \vdash \varphi \Leftrightarrow \Phi \models \varphi$.

4 Modal Logic as a Specification Language for Coalgebras

First, we define the modal language that we will use to specify coalgebras, we give a semantics and we show—using the examples from section 2—that it allows for an intuitive formalisation of properties. We also show that modal logic avoids equations between states in a natural way. Second, it is shown that formulas are invariant under morphisms and modally definable classes of coalgebras are related to final coalgebras.

4.1 Language, Semantics, Examples

We first need a modal language. The set of propositions and the set of modal operators are determined by the functor $F(S) = (B_1 + C_1 \times S)^{A_1} \times \ldots (B_n + C_n \times S)^{A_n}$ in the following way. For each component $(B_i + C_i \times S)^{A_i}$ and each $a \in A_i$ there is a modal operator $[i, a]$. And for each $d \in B_i + C_i$ we have an atomic proposition $(i, a) = d$ (read as "output of message i with argument value a is d"). In the case that A contains only one element we prefer to write simply $[i]$ and $i = d$. This gives a modal language $\mathcal{L}^F$.

As an illustration consider the example of the one-cell buffer of section 2.

(Recall that $n = 2$, $A_1 = A$ and A_2 a one element set.) Suppose we want to specify that a *store* into the empty buffer stores indeed, that a *store* into a full buffer has no effect and that a *read* empties the buffer. Then—writing $[store(a)]$ for $[1, a]$, $[read]$ for $[2]$, $read = d$ for $2 = d$, and *error* for the element of 1—we can formalise the conditions above as follows.

$$read = error \rightarrow [store(a)]read = a$$
$$read = a \rightarrow [store(b)]read = a$$
$$[read]read = error$$

Note that the above expressions are not strictly speaking formulas of our modal language. They are axiom schemes that yield formulas for all $a, b \in A$.

Having gained some intuition, here is the definition of the language and its semantics.

Definition 4.1 (The modal language $\mathcal{L}^F$) *Let F be a functor on* **Set** *of the form $F(S) = (B_1 + C_1 \times S)^{A_1} \times \ldots (B_n + C_n \times S)^{A_n}$. Then the set of atomic propositions* $\mathbf{P}$ *for F consists of propositions $(i, a) = d$ for all $1 \leq i \leq n, a \in A_i, d \in B_i + C_i$. The modal language obtained from* $\mathbf{P}$ *by adding the constant $\perp$, boolean connectives and modal operators $[i, a]$ for all $1 \leq i \leq n, a \in A_i$ is called $\mathcal{L}^F$. $\langle i, a\rangle\varphi$ is an abbreviation for $\neg[i, a]\neg\varphi$.*

Note that given the modal language we automatically have a semantics in terms of Kripke models as has been recalled in the previous section. Here we give a semantics of the language in terms of coalgebras. The two semantics are related in section 5.1.

Definition 4.2 (Semantics $\models^F$ of $\mathcal{L}^F$) *Let (S, f) be a F-coalgebra, $s \in S$, φ a formula of $\mathcal{L}^F$, and $F(S) = (B_1 + C_1 \times S)^{A_1} \times \ldots (B_n + C_n \times S)^{A_n}$. The semantics for boolean connectives is as usual. For propositions and modal operators:*

$$\begin{array}{lll} s \models^F (i, a) = d & \textit{iff} & (\pi_i \circ f)(s)(a) = d \textit{ or} \\ & & \textit{there is } t \in S \textit{ s.t. } (\pi_i \circ f)(s)(a) = (d, t) \\ s \models^F [i, a]\varphi & \textit{iff} & \textit{for all } c \in C_i, t \in S \ (\pi_i \circ f)(s)(a) = (c, t) \ \Rightarrow \ t \models^F \varphi \end{array}$$

Note that the "or" on the right hand side of the first clause corresponds to the "+" in $(B_i + C_i \times S)^{A_i}$.

The satisfaction relation $\models^F$ is always relative to a specific coalgebra (S, f). If we want to emphasise this, we write $(S, f), s \models^F \varphi$ (or sometimes simply $S, s \models^F \varphi$). For a theory of (S, f) in state s we write $\mathrm{Th}^F(S, s) = \{\varphi : S, s \models^F \varphi\}$.

Next, we want to show how to specify and prove properties of *newly created objects*. The natural way to do this in our approach is to add a predicate *New* to the language having the newly created states as its extension. Such a predicate may easily be expressed in the coalgebraic setting by considering the functor $\mathrm{Bool} \times F(S)$ instead of $F(S)$, where Bool denotes the set of truth values $\{\mathit{true}, \mathit{false}\}$. In other words, the characteristic function $S \to \mathrm{Bool}$ of *New* is added to the description of the class. This is seen to be compatible with the functors being of the form

$$F(S) = (B_1 + C_1 \times S)^{A_1} \times \ldots (B_n + C_n \times S)^{A_n}$$

since we may choose A_1 to be a one-element set, C_1 to be empty, and $B_1 = \mathrm{Bool}$. In the corresponding language $\mathcal{L}^F$ we write *new* to denote the atomic proposition $1 = \mathit{true}$. For an example consider the LIFO-queue below.

Before analysing more of the properties of this logic, we want to emphasise that modal logic gives us the appropriate language when we are not interested in particular states but only in states up to bisimulation. To make this clear let us specify the LIFO-queue from section 2. We first require

$$\begin{gathered}[\mathit{in}(a)]\mathit{out} = a \\ \mathit{New} \to \mathit{out} = \mathit{error}\end{gathered}$$

meaning that *out* yields a after input of a and that a newly created queue is empty. But something still lacks. How can we express that doing an *out* after an *in* gives us the "same" queue as before? Of course we do not want to say that we really get the same queue. Since states are not observable what should be said is: doing *out* after *in* gives us a queue that has the same *behaviour* as the queue before. This can be expressed in our formalism by writing down the following axiom scheme: [3]

$$[\mathit{in}(a)][\mathit{out}]\varphi \leftrightarrow \varphi$$

Note that *specifying the above property with equational logic forces us to use not only equations between attributes but between states* (cf. the discussion in [15]). *Modal logic avoids the direct access to states in a natural way.* That the above axiom scheme expresses indeed the intended constraint on the behaviours is implied by theorem 5.1.

Last, we want to make the connection to Rößiger [21]. In his remark 2.5 it is shown that for the functors considered here the two logics coincide. In the LIFO-example, disregarding the *new* (i.e. taking as functor $F(S) = S^A \times (1 + A \times S)$), the remaining two axioms from above become

$$\begin{gathered}[\times_1 a](\times_2 +_2 \times_1)a, \\ [\times_1 a][\times_2 +_2 \times_2]\varphi \leftrightarrow \varphi.\end{gathered}$$

[3] The scheme denotes the set of all its instances with formulas of $\mathcal{L}^F$ substituted for φ (and elements of A substituted for a).

4.2 Some Properties of the Logic

As to be expected, validity of formulas is invariant under morphisms and bisimulations:

Proposition 4.3 *Given two F-coalgebras and a morphism $h : (S, f) \to (T, g)$ we have:*

$$(S, f), s \models^F \varphi \iff (T, g), h(s) \models^F \varphi$$

Proof. The proof is the usual induction on the structure of the formulas. Let us look at two cases. Let $a \in A_i, c \in C_i$. First, φ is $(i, a) = c$: $S, s \models^F (i, a) = c$ iff $\exists s' \in S : (\pi_i \circ f)(s)(a) = (c, s')$ iff $\exists s' \in S : (\pi_i \circ Fh \circ f)(s)(a) = (c, h(s'))$ iff $\exists t' \in T : (\pi_i \circ g \circ h)(s)(a) = (c, t')$ iff $T, h(s) \models^F (i, a) = c$.

Second, φ is $[i, a]\psi$, $s' \in S, t' \in T$. Then $S, s \models^F [i, a]\psi$ iff $(\pi_i \circ f)(s)(a) = (c, s') \Rightarrow S, s' \models^F \psi$ iff $(\pi_i \circ f)(s)(a) = (c, s') \Rightarrow T, h(s') \models^F \psi$ iff $(\pi_i \circ Fh \circ f)(s)(a) = (c, h(s')) \Rightarrow T, h(s') \models^F \psi$ iff $(\pi_i \circ g \circ h)(s)(a) = (c, t') \Rightarrow T, t' \models^F \psi$ iff $T, h(s) \models^F \psi$. $\square$

Then as a corollary of the above proposition we get that if $(S, s) \simeq^F (T, t)$ then also $\mathrm{Th}^F(S, s) = \mathrm{Th}^F(T, t)$.

Next, let us take a look at modally definable classes of coalgebras. Let F be a functor as described above and $\Phi \subset \mathcal{L}^F$ a set of formulas. Consider the class

$$\mathrm{Mod}(\Phi) = \{(S, f) \in \mathbf{Set}_F : \text{ for all } s \in S, \varphi \in \Phi : (S, f), s \models^F \varphi\}$$

of (coalgebra-) models of Φ. Obviously, $\mathrm{Mod}(\Phi)$ gives rise to a full subcategory of $\mathbf{Set}_F$. And it is closely related to the coalgebra T_Φ that is defined as the largest subcoalgebra of the final coalgebra T whose carrier is contained in $\{t \in T : t \models^F \Phi\}$:[4] As shown by the next theorem, specifications in modal logic work by defining subcoalgebras of the final coalgebra.

Theorem 4.4 *Let $\Phi \subset \mathcal{L}^F$. Then $S \in \mathrm{Mod}(\Phi)$ iff there is a (necessarily unique) morphism $S \to T_\Phi$.*

Proof. "only if:" Let T be the final coalgebra in $\mathbf{Set}_F$ (that exists and is explicitly described in [15]). There is a unique morphism $h : S \to T$. Since $S \models \Phi$ the image of S is contained in the carrier of T_Φ, therefore h factors uniquely through T_Φ,[5] giving rise to a unique morphism $S \to T_\Phi$.
"if:" Immediate by definition of T_Φ and proposition 4.3. $\square$

A class of models K is said to be *closed under bisimulations* whenever from $S \in K$ and R a total bisimulation between S, S' it follows $S' \in K$. It is

[4] Existence of T_Φ follows from Theorem 6.4 in [22]
[5] Theorem 7.1 in [22]

called a covariety when it is closed under images, subcoalgebras and disjoint unions. We follow Gumm and Schröder [8] and call covarieties that are closed under bisimulations *complete covarieties*. With this definition we get as an immediate corollary:

Corollary 4.5 *Let* $\Phi \subset \mathcal{L}^F$*. Then* $\mathrm{Mod}(\Phi)$ *is a complete covariety.*

Proof. That $\mathrm{Mod}(\Phi)$ is a covariety follows from the theorem above and Rutten [22], theorem 15.1. Closure under bisimulations is an obvious consequence of proposition 4.3. $\Box$

Gumm and Schröder [8] analyse logics where the converse (i.e. any complete covariety is definable by a set of formulas) also holds. See the paragraphs following theorem 5.1 for a discussion on how this relates to our case.

5 Transferring Results from Modal Logic

One reason why it is nice to use modal logic as a specification language is that the theory of modal logics gives us a lot of tools to design appropriate logics, to prove results about these logics, and to work with the logics (interactive theorem provers, model checkers). To get access to this area we give a translation of coalgebras into Kripke models and show that the coalgebraic semantics of $\mathcal{L}^F$ coincides with Kripke semantics. This translation may then be used to transfer results from modal logic. In this paper it is used to show that $\mathcal{L}^F$ allows us to define behaviours up to bisimulation and to give a complete axiomatisation of $\mathcal{L}^F$. In connection with the completeness proof we also discuss the relationship of the canonical model for $\mathcal{L}^F$ and the final F-coalgebra.

5.1 The Translation

The logic $\mathcal{L}^F$ has been given a semantics in terms of coalgebras. On the other hand, like any modal logic in the style of section 3, it has also a semantics w.r.t. Kripke models. The connection between both is given by a translation of the category of F-coalgebras into the category of Kripke models for $\mathcal{L}^F$. It is a full and faithful embedding preserving and reflecting all interesting logical properties.

Let F be a functor on **Set** of the form $F(S) = (B_1 + C_1 \times S)^{A_1} \times \ldots (B_n + C_n \times S)^{A_n}$. The category K^F of F-Kripke models is given as follows. Let $\mathbf{P}$ be the set of atomic propositions for the functor F (see definition 4.1). Then a F-Kripke model is a Kripke model $(W, \mathcal{R}, V)$ where W is a set, $\mathcal{R}$ is a family $(R_{i,a})_{1 \le i \le n, a \in A_i}$ of relations and $V : \mathbf{P} \to \mathcal{P}(W)$ a mapping from propositions into the powerset of W. Since the methods are supposed to be functions we have the following restrictions on the relations and on the extensions of the propositions. For all $w \in W, 1 \le i \le n, a \in A_i, b \in B_i, c \in C_i$ it holds:

(i) $w \in V((i,a) = b) \;\Rightarrow\; w$ has no $R_{i,a}$-successor,

(ii) $w \in V((i,a) = c) \;\Rightarrow\; w$ has exactly one $R_{i,a}$-successor,

(iii) in w holds exactly one proposition of $\{(i,a) = d : d \in B_i + C_i\}$.

The morphisms in the category K^F are the p-morphisms (see section 3). Note that K^F is a full subcategory of the category of Kripke models for $\mathcal{L}^F$. In the following we will see that $\mathbf{Set}_F$ and K^F are equivalent (even isomorphic).

We define two functors $sk : \mathbf{Set}_F \to K^F$ and $ks : K^F \to \mathbf{Set}_F$ which will be isomorphisms that preserve in particular logical equivalence and bisimilarity.

$sk : \mathbf{Set}_F \to K^F$ maps every F-coalgebra (S, f) to the F-Kripke model $(S, \mathcal{R}, V)$ where

$$\begin{aligned}
(s,t) \in R_{i,a} \;&\Leftrightarrow\; \exists c \in C_i : (\pi_i \circ f)(s)(a) = (c,t) \\
s \in V((i,a) = b) \;&\Leftrightarrow\; (\pi_i \circ f)(s)(a) = b \\
s \in V((i,a) = c) \;&\Leftrightarrow\; \exists t \in S : (\pi_i \circ f)(s)(a) = (c,t)
\end{aligned}$$

$ks : K^F \to \mathbf{Set}_F$ maps every F-Kripke model $(S, \mathcal{R}, V)$ to the F-coalgebra (S, f) where

$$\begin{aligned}
(\pi_i \circ f)(s)(a) = b \;&\Leftrightarrow\; s \in V((i,a) = b) \\
(\pi_i \circ f)(s)(a) = (c,t) \;&\Leftrightarrow\; s \in V((i,a) = c) \text{ and } (s,t) \in R_{i,a}
\end{aligned}$$

On morphisms sk and ks are the identity.

sk and ks are isomorphisms that preserve all interesting properties. We need in particular:

(a) $\quad S, s \models^F \varphi \;\Leftrightarrow\; sk(S), s \models \varphi$

(b) $\quad (M, s) \simeq (N, t) \;\Rightarrow\; (ks(M), s) \simeq^F (ks(N), t)$

(c) $\quad ks \circ sk = id_{\mathbf{Set}_F}$

5.2 Logical equivalence implies bisimilarity

Logical equivalence implies bisimilarity. The proof uses the well-known fact that this property holds for image-finite Kripke models.

Theorem 5.1 $\mathrm{Th}^F(S, s) = \mathrm{Th}^F(T, t) \;\Rightarrow\; (S, s) \simeq^F (T, t)$.

Proof. Suppose $\mathrm{Th}^F(S, s) = \mathrm{Th}^F(T, t)$. Then $\mathrm{Th}(sk(S), s) = \mathrm{Th}(sk(T), t)$ by (a). Since $sk(S)$ and $sk(T)$ are so-called *image-finite* Kripke models, logical equivalence implies bisimilarity, that is, $(sk(S), s) \simeq (sk(T), t)$. By (b) and (c) we get $(S, s) \simeq^F (T, t)$. $\square$

The argument remains valid if we would allow the functors F to be built from the finite powerset functor. [6] More generally, the above argument is possible whenever the translation into Kripke models yields a class of models having the so-called Hennessy-Milner property. For detailed discussions of this concept see Goldblatt [7] and Hollenberg [12].

Note also that our expressiveness result is not strong enough to get the converse of corollary 4.5. To achieve this we would either need a "global" version of the above theorem saying $\mathrm{Th}^F(S) = \mathrm{Th}^F(T) \Rightarrow S \simeq^F T$ or a logic with infinite conjunctions and disjunctions. See Gumm and Schröder [8] for a proof in the latter case.

5.3 Axiomatisations

Our next aim is to give a complete axiomatisation of the logic $\mathcal{L}^F$. The strategy is to use again the translation of coalgebras into Kripke models. First we need to find the set Σ^F of axioms. If we then can show that $K^F \models \varphi \Rightarrow \Sigma^F \models \varphi$ we get a completeness result for $\models^F$ from the completeness result of modal logic. Unfortunately, to find a strong enough set Σ^F, the restriction to finite output sets B_i, C_i is needed. We give a complete axiomatisation for this restricted case using the canonical model for $\mathcal{L}^F$ and comment on the relation of the final coalgebra and the canonical model. Then we show why the restriction to finite output sets is needed, and sketch some possible ways to overcome this.

Let again F be given by $F(S) = (B_1 + C_1 \times S)^{A_1} \times \ldots (B_n + C_n \times S)^{A_n}$. To find Σ^F recall the definition of K^F in section 5.1. The restrictions on the models expressed there (see (i)-(iii)) have now to be formulated using modal formulas. Recall that $[i,a]\bot$ expresses that a world has no successor and that $[i,a]\varphi \to \langle i,a\rangle\varphi$ expresses that a world has at least one successor, cf. [5]. Furthermore $\langle i,a\rangle\varphi \to [i,a]\varphi$ expresses (on frames and also on the canonical model, see the proof of the theorem below) that every world has at most one successor. We therefore get the following axioms ((Ax1) corresponding to (i) and (Ax2),(Ax3) to (ii)):

(Ax1) $(i,a) = b \to [i,a]\bot$ for all $b \in B_i$

(Ax2) $(i,a) = c \to ([i,a]\varphi \to \langle i,a\rangle\varphi)$ for all $c \in C_i, \varphi \in \mathcal{L}^F$

(Ax3) $\langle i,a\rangle\varphi \to [i,a]\varphi$

Next we have to express that each method yields exactly one output value. At this point (see (Ax5)) we need that all the sets B_i, C_i are finite.

[6] For example functors described by $F{::=}C \mid Id \mid F+F \mid F \times F \mid F^C \mid \mathcal{P}_\omega$ where C a constant functor and $\mathcal{P}_\omega$ the finite covariant powerset functor.

(Ax4) $\quad (i,a) = d \to \neg(i,a) = d'$ for all $d \neq d', d, d' \in B_i + C_i$

(Ax5) $\quad \bigvee_{d \in B_i + C_i} (i,a) = d$

Let Σ^F be the set of $\mathcal{L}^F$-formulas defined by the five axiom schemes above.

Theorem 5.2 (Completeness for $\models^F$) *Let there be a functor on* **Set** $F(S) = (B_1 + C_1 \times S)^{A_1} \times \ldots (B_n + C_n \times S)^{A_n}$ *with all the* B_i, C_i *finite sets and let* φ *be a* $\mathcal{L}^F$*-formula and* Γ *be a set of* $\mathcal{L}^F$*-formulas. Then* $\Gamma \models^F \varphi \;\Leftrightarrow\; \Gamma \cup \Sigma^F \vdash \varphi$.

Proof. " $\Leftarrow$ " is a standard correctness proof. For " $\Rightarrow$ ", using the translation of section 5.1, it is enough to show that $\Gamma \cup \Sigma^F \not\vdash \varphi$ implies the existence of a model $M \in K^F : M \models \Gamma$ & $M \not\models \varphi$. We define $M^F = (W^F, \mathcal{R}^F, V^F)$ to be the canonical model [7] of $\Gamma \cup \Sigma^F$. That is, W^F is the set of maximal $(\Gamma \cup \Sigma^F)$-consistent sets of $\mathcal{L}^F$-formulas, $wR^F_{i,a}v \;\Leftrightarrow\; (\forall \psi \in \mathcal{L}^F : [i,a]\psi \in w \;\Rightarrow\; \psi \in v)$, $w \in V^F(p) \;\Leftrightarrow\; p \in w$. The canonical model has the property that $M^F, w \models \psi \;\Leftrightarrow\; \psi \in w$. Therefore $\Gamma \cup \Sigma^F \not\vdash \varphi$ implies that there is $w \in W$ such that $M^F, w \not\models \varphi$. Also, by construction $M^F \models \Gamma$. It remains to show that $M^F \in K^F$. That (Ax1,2,4,5) enforce the intended conditions on M^F should be obvious. That (Ax3) implies that any world in M^F has at most one $R^F_{i,a}$ successor is an easy exercise that may be found in [5]. $\square$

We have shown that the canonical model M^F for $\mathcal{L}^F$ is in K^F and hence can be thought of as a F-coalgebra. Is M^F also the final coalgebra? This is indeed the case (cf. Rößiger[21], theorem 7.1): given any coalgebra S define a function $h : S \to M^F$ via $f(s) = \mathrm{Th}(S, s)$. It is easy to show that h is well defined and satisfies the conditions (i) and (iii) of the definition of a p-morphism. For condition (ii) use that any world in M^F has at most one $R^F_{i,a}$ successor. Uniqueness of h is obvious. Hence M^F is final in K^F. But it is important to note that for more general functors the canonical model is *not* the final coalgebra. The reason is that the canonical model can be understood as the disjoint union of all models quotiented by logical equivalence whereas the final coalgebra is the quotient w.r.t. bisimulation which, generally, is a finer equivalence relation. Consequently, in cases where the logic is too weak to characterise worlds up to bisimulation, the existence of a morphism into the canonical model will fail (but if it exists it is unique).

Back to the completeness result, we have shown that if B_i, C_i are finite the calculus for $\mathbf{K}_{\mathcal{L}^\mathbf{F}}$ as presented in section 3 together with Σ^F as axioms is a strongly complete calculus for $\mathcal{L}^F$. We will call it $\mathbf{C}^F$.

How can we get rid of the restriction that the sets of output values B_i, C_i are finite? Possible ideas are to use modal predicate logic or to allow infinite conjunctions. But then it could be that completeness is lost. Here, we want to make a different proposal. It seems natural to allow that methods do not yield

[7] See e.g. Goldblatt [5] for details.

any output unless they are forced to do so by the axioms of the specification. In other words, the fact that methods are not partial is not any more expressed automatically by the syntax but has to be specified in the axiomatic part. We then do not need (Ax5) any more. This may seem to render the task of specifying more complicated but looking at our two examples from above we see that no changes to the axioms are necessary.

To be more explicit, things work out as follows. Define K^{F^-} as K^F but allowing models that have states where $\neg(i, a) = d$ holds for all $d \in B_i + C_i$. Define Σ^{F^-} as Σ^F but without (Ax5). We then can allow the sets B_i, C_i to be arbitrary. The proof of the theorem can be adopted almost literally. We therefore get a strongly complete calculus $\mathbf{C}^{F^-}$ for $\models^{F^-}$.

To summarise from a more practical point of view: Proofs that do not use (Ax5) yield formulas that are also valid for non-finite output sets. The above completeness result together with the examples (see below) indicates that the loss of (Ax5) is not important.

6 Proving Properties of Specifications

In this section we use the calculus for $\mathcal{L}^F$ as given in the preceding subsection to show some example proofs. Consider the LIFO-example from section 2 and the following valid formulas:

$$
\begin{array}{ll}
(*) & [in(a)]\neg out = error \\
(**) & [in(a_1)] \ldots [in(a_k)][out]^i out = a_{k-i},\ 0 \leq i < k \\
(***) & new \rightarrow [in(a_1)] \ldots [in(a_k)][out]^k out = error,\ 0 \leq k
\end{array}
$$

where $[out]^i$ is an abbreviation for i-times $[out]$. The meaning of (*) is that after input of a the queue is not empty. (**) says that after k inputs and $i < k$ outputs a further output yields the $(k-i)$-th input value. (***) states that starting with a newly created queue and making k inputs and then as many outputs gives back an empty queue.

We give derivations in the calculus $\mathbf{C}^{F^-}$. The last column tells how the current line was derived. For example, (prop)(dist)(nec)(2) in the derivation below means that we applied rule (nec) to the formula derived in line 2, then used axiom (dist) and then applied some propositional reasoning (in that case only (mp)) to derive the actual formula.

First for (*).

(1)	$[in(a)]out = a$	axiom from the spec
(2)	$out = a \to \neg out = error$	(Ax4) from Σ^{F^-}
(3)	$[in(a)]out = a \to [in(a)]\neg out = error$	(prop)(dist)(nec)(2)
(4)	$[in(a)]\neg out = error$	(mp)(1,3)

For (**) let us see how things work for the instance

$$[in(a_1)][in(a_2)][in(a_3)][in(a_4)][out][out]out = a_2.$$

The derivation is as follows:

(1)	$[in(a_4)][out][out]out = a_2 \leftrightarrow [out]out = a_2$	spec
(2)	$[in(a_3)][out]out = a_2 \leftrightarrow out = a_2$	spec
(3)	$[in(a_2)]out = a_2$	spec
(4)	$[in(a_2)][in(a_3)][out]out = a_2 \leftrightarrow [in(a_2)]out = a_2$	(prop)(dist)(nec)(2)
(5)	$[in(a_2)][in(a_3)][in(a_4)][out][out]out = a_2 \leftrightarrow$ $[in(a_2)][in(a_3)][out]out = a_2$	(prop)(dist)(nec)(1)
(6)	$[in(a_2)][in(a_3)][in(a_4)][out][out]out = a_2$	(prop)(3,4,5)
(7)	$[in(a_1)][in(a_2)][in(a_3)][in(a_4)][out][out]out = a_2$	(nec)(6)

The derivation of (***) follows the same idea.

7 Coalgebraic Logic as a Specification Language

So far, our treatment of specifications of objects and classes using modal logic was inspired by regarding them as coalgebras. But the results we proved were obtained by viewing coalgebras as Kripke models. In this respect, the general question underlying our approach is: to what extent can coalgebras be considered as Kripke models? Unfortunately, giving a uniform translation from coalgebras to transition systems for a larger variety of functors including exponentiation and powerset does seem to yield rather complicated Kripke models that do not give rise to logics naturally connected with the functor.

It is therefore natural to ask for a logic that depends in a canonical way on the functor and is thus truly coalgebraic. Such a logic has recently been developed by Moss [19]. We show examples of specifications in coalgebraic logic and give a translation from $\mathcal{L}^F$ into coalgebraic logic.

First the definition of coalgebraic logic from [19] is given. For a detailed discussion and results, the reader is referred to the original paper. We need the category **SET** of classes and set-continuous functions. The functors F are on **SET** and have to be set-based, standard and to preserve weak pullbacks. A functor F on **Set** can be extended (and will be where necessary without further mentioning) to a functor on **SET** by defining $FK = \bigcup\{FX : X \subset K, X \text{ a set}\}$ for classes K. The F-language $\mathcal{L}_F$ is defined to be the least class satisfying:

$$\begin{array}{lcl} \Phi \subset \mathcal{L}_F,\ \Phi \text{ a set} & \Longrightarrow & \bigwedge \Phi \in \mathcal{L}_F \\ \varphi \in \mathcal{L}_F & \Longrightarrow & \neg\varphi \in \mathcal{L}_F \\ \varphi \in F\mathcal{L}_F & \Longrightarrow & \varphi \in \mathcal{L}_F \end{array}$$

Due to the first clause $\mathcal{L}_F$ is a proper class. The last clause uses the fact that F is a functor on **SET** and as such can also be applied to classes of formulas. The second clause is not a proper part of $\mathcal{L}_F$ as defined in [19] but it is shown there that negation may be added. Here, negation is needed only to simplify the writing of specifications. Everything could be done using infinitary disjunctions instead. The latter are really needed: $\mathcal{L}_F$ without negation and disjunction is strong enough to characterise every state of a coalgebra up to bisimulation, but not strong enough to define classes of coalgebras.

Given a coalgebra (S, f) the semantics is given by the least relation $\models_F \subset S \times \mathcal{L}_F$ such that:

$$\begin{array}{lcl} s \models_F \varphi \text{ for all } \varphi \in \Phi, \Phi \subset \mathcal{L}_F,\ \Phi \text{ a set} & \Longrightarrow & s \models_F \bigwedge \Phi \\ s \not\models_F \varphi & \Longrightarrow & s \models_F \neg\varphi \\ \text{there is } w \in F(\models_F) \text{ s.t. } F\pi_1(w) = f(s), F\pi_2(w) = \varphi & \Longrightarrow & s \models_F \varphi \end{array}$$

The last clause makes use of $\models_F$ being in **SET** and applies F to it. How it works is best seen looking at some examples, see below. That $\models_F$ exists is shown in [19].

Now we can proceed to the examples. We do not consider here the axioms concerning the newly created objects. Consider the first three axioms of the one-cell buffer in section 4. In coalgebraic logic we can write them as ($*$ being the error message):

$$(true^A, *) \to (\lambda a.(true^A, (a, true)), *)$$

$$(true^A, (a, true)) \to (\lambda b.(true^A, (a, true)), (a, true))$$

$$\bigvee\{(true^A, z) : z \in 1 + A \times \{(true^A, *)\}\}$$

where $true^A$ is the constant function $A \to \{true\}$.

At first sight the main difference is that we have no direct access to the single components *store* and *read* (recall that *store* corresponds to the first component, *read* to the second). This is also the reason for the infinite disjunctions in the third clause.

Let us take a close look at the first clause. The premise $(true^A, *)$ tells that *read* yields *error* and specifies nothing about the *store*. The conclusion $(\lambda a.(true^A, (a, true)), *)$ says that we are in a state where the *store* is in accordance with $\lambda a.(true^A, (a, true))$ and the *read* yields *error*. Now, some thought shows that $\lambda a.(true^A, (a, true))$ means that storing a gives a state where $(true^A, (a, true))$ holds. And this formula describes exactly those states where *read* yields a. The reader is invited to check this, paying special attention to the third clause of the definition of $\models_F$.

The LIFO-example. The axioms become:

$$\bigvee_{z \in 1 + A \times \{true\}} (\lambda a.(true^A, (a, true)), z)$$
$$(\bigvee_{z \in 1 + A \times \{true\}} (\lambda a.(true^A, (a, \varphi)), z)) \leftrightarrow \varphi$$

Note that the infinite disjunctions are needed to express that the properties are independent from the first element in the queue. The next section shows that there is a way to give access to the single components and thereby eliminating the disjunctions from the specifying formulas and reintroducing the modal operators.

7.2 Translating Modal Logic into Coalgebraic Logic

Coalgebraic Logic gives a general way to get a logic for coalgebras. But one disadvantage is that it lacks the intuitive box and diamond operators of modal logic. Translating $\mathcal{L}^F$ into $\mathcal{L}_F$ means to render into coalgebraic logic the modal operators.

Definition 7.1 (Translation T from $\mathcal{L}^F$ to $\mathcal{L}_F$) *The boolean operators are translated in the obvious way. For propositions and modal operators (using $a, a' \in A_i, b \in B_i, c \in C_i, \varphi \in \mathcal{L}^F$):*

$$\begin{aligned}
T((i,a) = b) &= \bigvee \{(g_1, \dots g_n) : g_j \in A_j \to B_j + C_j \times \{true\}, g_i(a) = b\} \\
T((i,a) = c) &= \bigvee \{(g_1, \dots g_n) : g_j \in A_j \to B_j + C_j \times \{true\}, g_i(a) = (c, true)\} \\
T([i,a]\varphi) &= \bigvee \{(g_1, \dots g_n) : \; g_j \in A_j \to B_j + C_j \times \{true\} \textit{ for all } j \neq i, \\
&\qquad\qquad g_i(a) \in B_i + C_i \times \{T(\varphi)\}, \\
&\qquad\qquad g_i(a') \in B_i + C_i \times \{true\} \textit{ for all } a' \neq a\}
\end{aligned}$$

The next proposition gives a characterisation of the translation of $\langle i, a \rangle$.

Proposition 7.2

$$\models_F T(\langle i,a\rangle\varphi) \leftrightarrow \bigvee\{(g_1,\dots g_n) : g_j \in A_j \to B_j + C_j \times \{true\} \textit{ for all } j \neq i,\\ g_i(a) \in C_i \times \{T(\varphi)\},\\ g_i(a') \in B_i + C_i \times \{true\} \textit{ for all } a' \neq a\}$$

The next theorem states that specifications in the language $\mathcal{L}^F$ can also be considered as specifications in coalgebraic logic.

Theorem 7.3 *Let F be a functor on the category* **Set** *of the form $F(S) = (B_1 + C_1 \times S)^{A_1} \times \dots (B_n + C_n \times S)^{A_n}$, (S, f) a F-coalgebra and $\varphi \in \mathcal{L}^F$. Then*

$$\textit{for all } s \in S \ : \ s \models^F \varphi \iff s \models_F T(\varphi).$$

Proof. By induction on the structure of φ.

$\boldsymbol{(i,a) = b.}$ " $\Rightarrow$:" Suppose $s \models^F (i,a) = b$. We have to find w in $F(\models_F)$ such that $(F\pi_1)(w) = f(s)$ and $(F\pi_2)(w)$ a formula of the disjunction. Define w to be a tuple $(w_1, \dots w_n)$ such that for all $1 \leq j \leq n$, $w_j : A_j \to B_j + C_j \times (S \times \mathcal{L}_F)$ s. t.

$$w_j(a) = \begin{cases} b & \text{if } (\pi_j \circ f)(s)(a) = b \\ (c, (t, true)) & \text{if } (\pi_j \circ f)(s)(a) = (c, t) \end{cases}$$

" $\Leftarrow$:" Suppose there is a $w \in F(\models_F)$ such that $(F\pi_1)(w) = f(s)$ and $(F\pi_2)(w) = g$ for a formula g of the disjunction. Then $g = (g_1, \dots g_n)$ and $g_i(a) = b$. Therefore $(\pi_i \circ f)(s)(a) = (\pi_i \circ F\pi_1)(w)(a) = b$.

$\boldsymbol{(i,a) = c.}$ Similar argument.

$\boldsymbol{[i,a]\psi.}$ " $\Rightarrow$:" Choose $w = (w_1, \dots w_n)$ such that for all $1 \leq j \leq n, j \neq i$

$$w_j(a) = \begin{cases} b & \text{if } (\pi_j \circ f)(s)(a) = b \\ (c, (t, true)) & \text{if } (\pi_j \circ f)(s)(a) = (c, t) \end{cases} \quad \text{and}$$

$$w_i(a) = \begin{cases} b & \text{if } (\pi_i \circ f)(s)(a) = b \\ (c, (t, T(\psi))) & \text{if } (\pi_i \circ f)(s)(a) = (c, t) \end{cases}$$

As above $F\pi_1(w) = f(s)$ and $F\pi_2(w)$ is a formula of the disjunction. It remains to show that w is indeed in $F(\models_F)$. This follows from the definition of $\models^F$ and the induction hypothesis.

" $\Leftarrow$:" Suppose $(\pi_i \circ f)(s)(a) = (c, t)$. We have to show $t \models^F \psi$. Let g be a formula of the disjunction $T([i,a]\psi)$ with $s \models_F g$. It follows that there is a $w \in F(\models_F)$ s.t. $F\pi_1(w) = f(s), F\pi_2(w) = g$. Because of $(\pi_i \circ f)(s)(a) = (c, t)$ and the definition of g we get $(\pi_i \circ F\pi_1(w))(a) = (c, t)$ and $(\pi_i \circ F\pi_2(w))(a) = (c, T(\psi))$, hence $(\pi_i \circ w)(a) = (c, (t, T(\psi)))$. Therefore $t \models_F T(\psi)$. By induction hypothesis it follows $t \models^F \psi$. $\square$

8 Coalgebras, Modal Logic, and Object Orientation

The modal logic presented in this paper was designed to show that modal logics provide a natural language to speak about coalgebras. It was not the aim to propose a language capable of specifying fully fledged object oriented systems. This section discusses possibilities and problems of developments in this direction.

First let us hint at possible extensions of our logic dealing with the issues of verification, temporal specifications and inheritance. Concerning verification, our formalism only allows to specify properties of methods but not to verify correctness of implementations. This could be achieved by extending our logic by features of propositional dynamic logic (PDL) [8] as follows. PDL is a modal logic that has modal operators $[\alpha]$ for each statement α of a given imperative programming language. As in our logic, the intended meaning of a modal formula $[\alpha]\varphi$ is: after all executions of the statement α proposition φ holds. In addition, PDL has algebraic operations on the modalities corresponding to the operations allowing to form statements. For example $[\alpha], [\beta]$ being modalities of PDL, there is also a modality $[\alpha; \beta]$, corresponding to the composition operator ";" in the programming language. The intended meaning of the operations on the modalities is expressed by certain axioms (for example $[\alpha; \beta]\varphi \leftrightarrow [\alpha][\beta]\varphi$). In this way, using PDL as a logic on the level of the programming language and thinking of the formulas of our logic as special PDL formulas, we could use the (complete) calculus for PDL to verify that some implementation meets the specifications written in our logic.

We might also be interested in extending our logic with modal operators that allow to specify safety and liveness properties. In principle, the coalgebras being isomorphic to Kripke models, it is straight forward to use any of the many temporal logics designed for Kripke models like linear temporal logic or CTL*. For example, we may add to our logic two operators $\bigcirc$ and $\Box$ and interpret $\bigcirc$ by [9] $R_{\bigcirc} = \bigcup_{i,a} R_{i,a}$ and $R_{\Box}$ by the reflexive and transitive closure of $R_{\bigcirc}$. Hence $\bigcirc\varphi$ means "whatever is the method to be executed next, after its execution φ will hold" and $\Box$ is the corresponding always operator. But there is a technical problem due to the possibility of infinitely many modal operators $[i, a]$: To get a reasonable calculus we need to be able to infer $\bigcirc\varphi$ from the infinitely many premises $\{[i, a]\varphi$: for all $i, a\}$. Admitting a calculus with infinitary rules (but no infinitary formulas) and using as an essential ingredient the infinitary rule $\{[i, a]\varphi : \text{for all } i, a\} \vdash \bigcirc\varphi$, it is possible to give a complete calculus for a temporal logic for coalgebras with operators $\bigcirc$ and $\Box$. The completeness proof uses the technique developed in Segerberg [23] (but see also Goldblatt [6]).

[8] For more information on PDL see e.g. Goldblatt [5].

[9] Recall from section 5.1 that the $R_{i,a}$ are the relations interpreting the operators $[i, a]$.

How reasoning about inheritance could be integrated into our framework remains to be investigated. One approach that seems promising is to build on work by Uustalu [24] who gives modal logics to treat various aspects of inheritance.

Having discussed possibilities to deal with verification, temporal specifications, and inheritance we come to the main point concerning coalgebras and the OO-paradigm. It is still an open question how to deal with concurrency and communication inside the coalgebraic approach. Note that this point is also related to the question of how to model shared objects: Once we have a communication mechanism shared objects can be treated just as objects linked to the owners by message passing. And vice versa, shared objects could be used to model communication.

In order to further illustrate the issue we compare the coalgebraic approach with the one using distributed temporal logic (DTL), see [4] for an overview and further references and [9] for using DTL to give a complete semantics to the object oriented specification language TROLL [2]. The DTL approach is based on the view of objects as autonomous sequential agents capable of synchronous communication. Each object is modelled by a labelled sequential event structure. Synchronous communication is modelled by objects sharing the same event. Formulas of the logic are interpreted only locally w.r.t. a specific object. But formulas allow to express that an object shares an event with another object. For example, a formula local to object i may specify certain properties that have to hold for a different object j at the time when a common event is shared between i and j. Also, treating actions like transactions in database theory it is possible to give a logic invariant under change of granularity and thus supporting refinement proofs [3].

We think that the coalgebraic approach and the object-as-agent approach are in some sense orthogonal. The coalgebraic approach emphasises classes, e.g. it leads to canonical notions of invariance and bisimulation for a class. Using PVS, it has been used by Jacobs and others [16,10,17] to implement verification tools for (parts of) Java. The features handled are mainly those related to the class structure of the system like inheritance, overriding, and late binding. Another way to put it is to say that the coalgebraic approach as understood today is well suited to deal with the static features of an object oriented system [10] but it is not clear how to integrate dynamic aspects like communication and concurrency. This is also highlighted by the work of Cenciarelli et al. [1] who give an event based semantics of Java dealing with threads and concurrency. Whereas this semantics is conceptually not too far away from the DTL approach it has, up to now, resisted all efforts to combine it with a coalgebraic approach.

[10] Late binding may be considered a dynamic concept but it can be handled in a static way, see [13].

The object-as-agent approach emphasises objects. It allows for a natural treatment of communication and concurrency but seems to be less suited to deal with aspects related to classes. Also it has not yet led to verification tools.

One way to combine both worlds has already been presented in Reichel [20]. We think it is promising to continue this line of research and to try to develop a logical framework allowing to deal with both worlds in coherent way. There is the possibility that modal logics may play a major role in it, allowing to be used for different purposes as describing imperative programs, specifying objects and classes, communication and concurrency. As mentioned already, this would give us the very powerful machinery of modal logic at hand, with all its results concerning completeness, definability, decidability. And—important for specifying and verifying programs—with all its tools like interactive theorem provers and model checkers.

9 Conclusion

We have seen that by conceiving of coalgebras as Kripke models modal logic may be used as a logic for coalgebras. It just has been sketched what may be achieved by this approach. The kind of the functors considered should be extended to include at least the powerset functor. Concerning the specification of classes and objects it would be interesting to include temporal reasoning allowing for the specification of safety and liveness properties. Further topics include inheritance, refinement, compositionality, communication and verification. Also extensions of the logic by quantification or infinitary conjunctions should be considered. More general questions concern the relation between equational and modal specifications. And whether there is a way to conceive of modal logic as a dual of equational logic (cf. the duality between modal algebras and Kripke models).[11]

I hope to have shown that modal logic may contribute interesting tools and results as well as some new questions to the theory of coalgebras.

Acknowledgement

I want to thank Martin Rößiger for the good collaboration in preparing the version of the paper for this issue. I am also indebted to Stephan Merz, Dirk Pattinson, and Alexander Knapp for careful reading of previous drafts.

[11] The proof of a Birkhoff-style co-variety theorem for modal logic in [18] supports indeed the slogan that modal logic is co-equational: The idea is that modal formulas specify images of morphisms in much the same way as equations specify kernels of morphisms (i.e. equivalence relations). Dualising Birkhoff's proof then yields that covarieties are the modally definable classes of coalgebras.

Without the regular discussions on coalgebras and object orientation with Alexander this paper would not exist. Last but not least I am grateful to the anonymous referees for their help and their questions.

References

[1] Pietro Cenciarelli, Alexander Knapp, Bernhard Reus, and Martin Wirsing. An Event-Based Structural Operational Semantics of Multi-Threaded Java. In Jim Alves-Foss, editor, *Formal Syntax and Semantics of Java*, volume 1523 of *Lect. Notes Comp. Sci.*, pages 157–200. Springer, Berlin, 1999.

[2] G. Denker and P. Hartel. TROLL—An Object Oriented Formal Method for Distributed Information System Design: Syntax and Pragmatics (TROLL Version 3.0). Technical Report Informatik Berichte 97-03, Technische Universität Braunschweig, 1997.

[3] G. Denker, J. Ramos, C. Caleiro, and A. Sernadas. A linear temporal logic approach to objects with transactions. In Michael Johnson, editor, *Algebraic Methodology and Software Technology (AMAST'97)*, volume 1349 of *Lecture Notes in Computer Science*, pages 170–184, 1997.

[4] H.-D. Ehrich, C. Caleiro, A. Sernadas, and G. Denker. Logics for Specifying Concurrent Information Systems. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 167–198. Kluwer, 1998.

[5] Robert Goldblatt. *Logics of Time and Computation*, volume 7 of *CSLI Lecture Notes*. Center for the Study of Language and Information, Stanford University, 1992. Second Edition.

[6] Robert Goldblatt. A framework for infinitary modal logic. In *Mathematics of Modality*, chapter 9. Center for the Study of Language and Information, Stanford University, 1993.

[7] Robert Goldblatt. Saturation and the Hennessy-Milner property. In A. Ponse et.al., editor, *Modal Logic and Process Algebra*, volume 53 of *CSLI Lecture Notes*. Center for the Study of Language and Information, Stanford University, 1995.

[8] H. Peter Gumm and Tobias Schröder. Covarieties and complete covarieties. In B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors, *Coalgebraic Methods in Computer Science (CMCS'98)*, volume 11 of *Electronic Notes in Theoretical Computer Science*, 1998.

[9] Peter Hartel. *Konzeptionelle Modellierung von Informationssystemen als verteilte Objektsysteme*. Reihe DISDBIS. infix-Verlag, Sankt Augustin, 1997.

[10] U. Hensel, M. Huisman, B. Jacobs, and H. Tews. Reasoning about classes in object-oriented languages: Logical models and tools. In Ch. Hankin (ed),

editor, *European Symposium on Programming*, volume 1381 of *Lecture Notes in Computer Science*, pages 105–121. Springer, Berlin, 1998.

[11] U. Hensel and H. Reichel. Defining equations in terminal coalgebras. In E. Astesiano, G. Reggio, and A.Tarlecki, editors, *Recent Trends in Data Type Specification (COMPASS/ADT 1995)*, volume 906 of *Lecture Notes in Computer Science*, pages 307–318, 1995.

[12] Marco Hollenberg. Hennessy-Milner classes and process algebra. In A. Ponse, M. de Rijke, and Y. Venema, editors, *Modal Logic and Process Algebra*, volume 53 of *CSLI Lecture Notes.* Center for the Study of Language and Information, Stanford University, 1995.

[13] B. Jacobs, J. van den Berg, M. Huisman, M. van Berkum, U. Hensel, and H. Tews. Reasoning about Java classes (preliminary report). *Object-Oriented Programming Systems, Languages and Applications, ACM Press*, pages 329–340, 1998.

[14] Bart Jacobs. Mongruences and cofree coalgebras. In V. S. Alagar and M. Nivat, editors, *Algebraic Methods and Software Technology (AMAST'95)*, volume 936 of *Lecture Notes in Computer Science*, pages 245–260, 1995.

[15] Bart Jacobs. Objects and classes, co-algebraically. In B. Freitag, C. B. Jones, C. Lengauer, and H.-J. Schek, editors, *Object-Orientation with Parallelism and Persistence*, pages 83–103. Kluwer Acad. Publ., 1996.

[16] Bart Jacobs. Coalgebraic reasoning about classes in object-oriented languages. In B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors, *Coalgebraic Methods in Computer Science (CMCS'98)*, volume 11 of *Electronic Notes in Theoretical Computer Science*, pages 235–246, 1998.

[17] Bart Jacobs. Coalgebras in specification and verification for object-oriented languages. *Newsletter 3 of the Dutch Association for Theoretical Computer Science (NVTI)*, pages 15–27, 1999.

[18] Alexander Kurz. A co-variety-theorem for modal logic. In *Proceedings of Advances in Modal Logic 2, Uppsala, 1998.* Center for the Study of Language and Information, Stanford University, 1999.

[19] Lawrence Moss. Coalgebraic logic. Submitted, 1997.

[20] Horst Reichel. An approach to object semantics based on terminal co-algebras. *Mathematical Structures in Computer Science*, 5(2):129–152, June 1995.

[21] Martin Rößiger. From modal logic to terminal coalgebras. This number.

[22] Jan Rutten. Universal coalgebra: A theory of systems. Technical Report CS R 9652, CWI, Amsterdam, 1996.

[23] Krister Segerberg. A model existence theorem in infinitary propositional modal logic. *Journal of Philosophical Logic*, 23:337–367, 1994.

[24] Tarmo Uustalu. Combining object-oriented and logic paradigms: A modal logic programming approach. In O. Lehrmann Madsen, editor, *Proceedings 6th European Conf. on Object-Oriented Programming, ECOOP'92*, volume 615 of *LNCS*, pages 98–113, Berlin, 1992. Springer.