

# Coalgebraic Modal Logic in CoCASL

Lutz Schröder and Till Mossakowski

Department of Computer Science, University of Bremen, and  
DFKI Lab Bremen, Germany

**Abstract.** We extend the algebraic-coalgebraic specification language CoCASL by full coalgebraic modal logic based on predicate liftings for functors. This logic is more general than the modal logic previously used in CoCASL and supports the specification of a variety of modal logics, such as graded modal logic, majority logic, and probabilistic modal logic. CoCASL thus becomes a modern modal language that covers a wide range of Kripke and non-Kripke semantics of modal logics via the coalgebraic interpretation.

## Introduction

The algebraic-coalgebraic specification language CoCASL [14] combines the algebraic specification of functional aspects of software with the specification of reactive systems following the emerging coalgebraic paradigm [21]. As a specification logic, modal logic plays an analogous role for coalgebra as equational logic does for algebra; in particular, modal logic respects the behavioural encapsulation of the state space.

Two notions of modal logic have been included in the original design of CoCASL. The first one treats observer operations that have a non-observable result sort as modalities (here, the sorts in the local environment are regarded to be the observable ones). The second notion of modal logic is based on a specific way of extracting modalities from datatypes with equations.

Recently, a more general formulation of *coalgebraic modal logic*, based on a notion of *predicate lifting*, has been proposed by Pattinson [18]. A substantial body of theory has developed for this generic logic, including results on duality, expressivity, decidability, and complexity [22, 9, 23, 26]. We hence propose to extend the CoCASL design by support for coalgebraic modal logic, including polyadic modal operators needed in order to obtain expressiveness in the general case, in particular for composite functors [22]. To this end, it is convenient to promote functors, which feature only implicitly in the original design of CoCASL, to first-class citizens. Specifically, functors can be defined as algebraic or coalgebraic datatypes, with the dependency on type arguments recorded explicitly; such functors can then be used as signatures for coalgebraic types. Predicate liftings for given functors can be specified without polymorphic axioms using a classification result from [22]; one thus obtains customized modal logics for user-defined system types. In particular, it becomes possible to specify e.g. graded or probabilistic modal operators.

## 1 Coalgebraic Modal Logic

We briefly recapitulate the basics of the coalgebraic interpretation of modal logic.

**Definition 1.** [21] Let  $T : \mathbf{Set} \rightarrow \mathbf{Set}$  be a functor, referred to as the *signature functor*, where  $\mathbf{Set}$  is the category of sets. A  $T$ -coalgebra  $A = (X, \xi)$  is a pair  $(X, \xi)$  where  $X$  is a set (of *states*) and  $\xi : X \rightarrow TX$  is a function called the *transition function*. A *morphism*  $(X_1, \xi_1) \rightarrow (X_2, \xi_2)$  of  $T$ -coalgebras is a map  $f : X_1 \rightarrow X_2$  such that  $\xi_2 \circ f = Tf \circ \xi_1$ .

We view coalgebras as generalised transition systems: the transition function assigns to each state a structured set of successors and observations.

Coalgebraic modal logic in the form considered here has been introduced as a specification logic for coalgebraically modelled reactive systems in [18], generalising previous results [8, 20, 10, 17]. The coalgebraic semantics is based on predicate liftings; here, we consider the notion of polyadic predicate lifting introduced in [22].

**Definition 2.** An  $n$ -ary finitary predicate lifting ( $n \in \mathbb{N}$ ) for a functor  $T$  is a natural transformation

$$\lambda : 2^n \rightarrow 2 \circ T^{\mathbf{OP}},$$

where  $2$  denotes the contravariant powerset functor  $\mathbf{Set}^{\mathbf{OP}} \rightarrow \mathbf{Set}$ , and  $2^n$  refers to its  $n$ -fold cartesian product.

A coalgebraic semantics for a modal logic consists of a signature functor and an assignment of a predicate lifting to every modal operator; we write  $[\lambda]$  for a modal operator that is interpreted using the lifting  $\lambda$ . Thus, a set  $A$  of finitary predicate liftings for  $T$  determines the syntax of a modal logic  $\mathcal{L}(A)$ . Formulae  $\phi, \psi \in \mathcal{L}(A)$  are defined by the grammar

$$\phi ::= \perp \mid \phi \wedge \psi \mid \neg\phi \mid [\lambda](\phi_1, \dots, \phi_n),$$

where  $\lambda$  ranges over  $A$  and  $n$  is the arity of  $\lambda$ . Disjunctions  $\phi \vee \psi$ , truth  $\top$ , and other boolean operations are defined as usual.

The satisfaction relation  $\models_C$  between states  $x$  of a  $T$ -coalgebra  $C = (X, \xi)$  and  $\mathcal{L}(A)$ -formulae is defined inductively, with the usual clauses for the boolean operations. The clause for the modal operator  $[\lambda]$  is

$$x \models_C [\lambda](\phi_1, \dots, \phi_n) \iff \xi(x) \in \lambda(\llbracket \phi_1 \rrbracket_C, \dots, \llbracket \phi_n \rrbracket_C)$$

where  $\llbracket \phi \rrbracket_C = \{x \in X \mid x \models_C \phi\}$ . We drop the subscripts  $C$  when  $C$  is clear from the context.

**Remark 3.** Coalgebraic modal logic exhibits a number of pleasant properties and admits non-trivial metatheoretic results, e.g. the following.

1. States  $x$  and  $y$  in  $T$ -coalgebras  $A$  and  $B$ , respectively, are called *behaviourally equivalent* if there exists a coalgebra  $C$  and morphisms  $f : A \rightarrow C$ ,  $g : B \rightarrow C$  such that  $f(x) = g(y)$ . It is easy to see that coalgebraic modal logic is *adequate*, i.e. invariant under behavioural equivalence. Thus, coalgebraic modal logic automatically ensures encapsulation of the state space.

2. Conversely, it is shown in [18, 22] that if  $T$  is  $\omega$ -accessible and  $A$  is separating in the sense that  $t \in TX$  is determined by the set  $\{(\lambda, A) \in \Lambda \times \mathcal{P}(X) \mid t \in \lambda(A)\}$ , then  $\mathcal{L}(A)$  is *expressive*, i.e. if two states satisfy the same  $\mathcal{L}(A)$ -formulae, then they are behaviourally equivalent.
3. As a consequence of 1., classes of coalgebras restricted by modal axioms have final models [11, 14].
4. The standard duality theory of modal logic, including the theory of ultrafilter extensions and bisimulation-somewhere-else, generalizes to coalgebraic modal logic [9].
5. Coalgebraic modal logic has the finite and shallow model properties [23, 26].
6. There are generic criteria for a coalgebraic modal logic to be (effectively) decidable [23, 26].

Coalgebraic modal logic subsumes a wide variety of modal logics (recall that a modal operator  $\Box$  is called *monotone* if it satisfies  $\Box(p \wedge q) \rightarrow \Box p$ , and *normal* if it satisfies  $(\Box p \wedge \Box q) \leftrightarrow \Box(p \wedge q)$ ).

**Example 4.** [18, 3, 23]

1. Let  $\mathcal{P}$  be the covariant powerset functor. Then  $\mathcal{P}$ -coalgebras are graphs, thought of as transition systems or Kripke frames. The predicate lifting  $\lambda$  defined by

$$\lambda_X(A) = \{B \subset X \mid B \subset A\}$$

gives rise to the standard box modality  $\Box = [\lambda]$ . This translates verbatim to the finitely branching case, captured by the finite powerset functor  $\mathcal{P}_{fin}$ .

2. Coalgebras for the functor  $N = 2 \circ 2^{\mathcal{P}}$  (composition of the contravariant powerset functor with itself) are neighbourhood frames, the canonical semantic domain of non-normal logics [2]. The coalgebraic semantics induced by the predicate lifting  $\lambda$  defined by

$$\lambda_X(A) = \{\alpha \in N(X) \mid A \in \alpha\}$$

is just the neighbourhood semantics for  $\Box = [\lambda]$ .

3. Similarly, coalgebras for the subfunctor  $\mathbf{Up}\mathcal{P}$  of  $N$  obtained by restricting  $N$  to upwards closed subsets of  $2^X$  are monotone neighbourhood frames [5]. Putting  $\Box = [\lambda]$ , with  $\lambda$  as above, gives the standard interpretation of the  $\Box$ -modality of monotone modal logic.
4. It is straightforward to extend a given coalgebraic modal logic for  $T$  with a set  $U$  of *propositional symbols*. This is captured by passing to the functor  $T'X = TX \times \mathcal{P}(U)$  and extending the set of predicate liftings by the liftings  $\lambda^a$ ,  $a \in U$ , defined by

$$\lambda_X^a(A) = \{(t, B) \in TX \times \mathcal{P}(U) \mid a \in B\}.$$

Since  $\lambda^a$  is independent of its argument, we can write the propositional symbol  $a$  in place of  $[\lambda^a]\phi$ , with the expected meaning.

5. The *finite multiset* (or *bag*) functor  $\mathcal{B}_{\mathbb{N}}$  maps a set  $X$  to the set of maps  $b : X \rightarrow \mathbb{N}$  with finite support. The action on morphisms  $f : X \rightarrow Y$  is given by  $\mathcal{B}_{\mathbb{N}}f : \mathcal{B}_{\mathbb{N}}X \rightarrow \mathcal{B}_{\mathbb{N}}Y, b \mapsto \lambda y. \sum_{f(x)=y} b(x)$ . Coalgebras for  $\mathcal{B}_{\mathbb{N}}$  are directed graphs with  $\mathbb{N}$ -weighted edges, often referred to as *multigraphs* [4], and provide a coalgebraic semantics for *graded modal logic* (GML): One defines a set of predicate liftings  $\{\lambda^k \mid k \in \mathbb{N}\}$  by

$$\lambda_X^k(A) = \{b : X \rightarrow \mathbb{N} \in \mathcal{B}_{\mathbb{N}}(X) \mid \sum_{a \in A} b(a) > k\}.$$

The arising modal operators are precisely the modalities  $\diamond_k$  of GML [4], i.e.  $x \models \diamond_k \phi$  iff  $\phi$  holds for more than  $k$  successor states of  $x$ , taking into account multiplicities. Note that  $\square_k$ , defined as  $\neg \diamond_k \neg$ , is monotone, but fails to be normal unless  $k = 0$ . A non-monotone variation of GML arises when negative multiplicities are admitted.

6. The *finite distribution functor*  $D_{\omega}$  maps a set  $X$  to the set of probability distributions on  $X$  with finite support. Coalgebras for the functor  $T = D_{\omega} \times \mathcal{P}(U)$ , where  $U$  is a set of propositional symbols, are probabilistic transition systems (also called *probabilistic type spaces* [7]) with finite branching degree. The natural predicate liftings for  $T$  consists of the propositional symbols (Item 4 above) together with the liftings  $\lambda^p$  defined by

$$\lambda^p(A) = \{P \in D_{\omega}X \mid PA \geq p\}$$

where  $p \in [0, 1] \cap \mathbb{Q}$ . The induced operators are the modalities  $L_p = [\lambda^p]$  of *probabilistic modal logic* (PML) [12, 7], where  $L_p \phi$  reads ‘ $\phi$  holds in the next step with probability at least  $p$ ’.

7. Let  $T$  be the functor given by  $TX = \mathcal{P}(X)^{(2^X)}$  (with  $\mathcal{P}$  the covariant powerset functor and  $2$  the contravariant powerset functor  $X \mapsto 2^X$ ). Then a  $T$ -coalgebra is a *standard conditional model* [2]. The strict implication operator  $\Rightarrow$  of *conditional logic* is interpreted using the binary predicate lifting  $\lambda$  defined by

$$\lambda_X(A, B) = \{f : 2^X \rightarrow \mathcal{P}(X) \mid f(A) \subset B\}.$$

The following simple fact gives immediate access to all predicate liftings that a functor admits, and will serve as a means of specifying predicate liftings in CoCAsL without introducing polymorphic axioms.

**Proposition 5.** [22] *For  $n \in \mathbb{N}$ ,  $n$ -ary predicate liftings for  $T$  are in bijective correspondence with subsets of  $T(2^n)$ , where  $2 = \{\top, \perp\}$ . The correspondence works by taking a predicate lifting  $\lambda$  to  $\lambda_{2^n}(\pi_1^{-1}\{\top\}, \dots, \pi_n^{-1}\{\top\}) \subseteq T(2^n)$ , where  $\pi_i : 2^n \rightarrow 2$  is the  $i$ -th projection, and, conversely,  $C \subseteq T(2^n)$  to the  $n$ -ary predicate lifting  $\lambda^C$  defined by*

$$\lambda_X^C(A_1, \dots, A_n) = (T(\chi_{A_1}, \dots, \chi_{A_n}))^{-1}[C]$$

for  $A_i \subseteq X$  ( $i = 1, \dots, n$ ), where angle brackets denote tupling of functions and  $\chi_A : X \rightarrow 2$  is the characteristic function of  $A \subseteq X$ .

## 2 Algebraic-Coalgebraic Specification

The algebraic-coalgebraic specification language CoCASL has been introduced in [14] as an extension of the standard algebraic specification language CASL. For the basic CASL syntax, the reader is referred to [1, 15]. We briefly explain the CoCASL features relevant for the understanding of the present work using the example specification shown in Fig. 1.

```

spec FAIRSTREAM =
  sort Unit
  •  $\forall x, y : Unit . x = y$ 
  sort Elem
  op  $c : Elem$ 
  then cofree {
    cotype Stream ::= ( $hd : ?Elem; tl : ?Stream$ ) | ( $stop : ?Unit$ )
    •  $\langle tl^* \rangle hd = c$ 
  }
end

```

**Fig. 1.** Specification of a fairness property.

Dually to CASL’s datatype construct **type**, CoCASL offers a **cotype** construct which defines coalgebraic process types; it is formally proved in [14] that one can indeed define for each cotype signature a functor  $T$  such that models of the cotype correspond to  $T$ -coalgebras. A simple example is the cotype *Stream* defined in Fig. 1. We first introduce a singleton type *Unit* and a (loosely interpreted) sort *Elem* with a constant  $c$ . Then, possibly terminating streams are defined using a cotype with two alternatives. Like a type declaration, a cotype declaration is essentially just a short way of declaring operations; specifically, the declaration of *Stream* produces *observer* operations  $hd : Stream \rightarrow ? Elem$ ,  $tl : Stream \rightarrow ? Stream$ , and  $stop : Stream \rightarrow ? Unit$ , with additional conditions on the definedness of observers which guarantee that models of the cotype *Stream* are essentially coalgebras for the functor  $\lambda X. Elem \times X + 1$ .

CoCASL’s modal logic now turns the observers  $hd$  and  $stop$  into flexible constants, as they have result sorts which stem from the local environment and hence are regarded as observable, while the observer  $tl$  has a result sort which is regarded as non-observable and hence induces modalities  $[tl]$  and  $\langle tl \rangle$ . The modality  $[tl]$  is interpreted as ‘for the tail of the stream, if any, it is the case that ...’, while  $\langle tl \rangle$  means ‘the stream has a tail, which satisfies ...’. The latter is slightly stronger, since it enforces that the tail is defined.

Modalities can be starred; this refers to states reachable by a finite number of iterated applications of an observer. The specification FAIRSTREAM of Fig. 1

hence expresses that all streams will always eventually output a  $c$  (before they possibly end). Here, the ‘always’ stems from the fact that by stating the modal formula, we mean that it holds on the cotype  $Stream$ , i.e. for all elements of type  $Stream$ .

The keyword **cofree** in the specification FAIRSTREAM further restricts the models to those that are final (over their  $Elem$ -part). In particular, one has a coinduction principle for  $Stream$ , and all possible behaviours are realised by the cotype  $Stream$  — i.e. up to isomorphism, the cotype  $Stream$  consists of all streams satisfying the modal axiom.

This form of specification only supports (coalgebras for) polynomial functors. A more general form of coalgebras involves *structured observations*: e.g. for non-deterministic automata, in each state, a *set* of successor states can be observed. In Fig. 2, finite sets are specified using a *free* type. Note that without structured observers, there can only be one or no successor state for a given state, while now we have a finite set of successor states. Hence, the modal logic needs to be adapted accordingly. As before, each observer with a (possibly structured) non-observable result leads to a modality. Since the observer  $next$  is additionally parameterized over an input sort  $In$ , we have modalities  $[next(i)]$  and  $\langle next(i) \rangle$  for  $i : In$ . The interpretation of these modalities is ‘after reading  $i$  in the current state, for *all* successor states, it is the case that ...’ and ‘after reading  $i$  in the current state, for *some* successor state, it is the case that ...’, respectively. E.g. the specification of non-deterministic automata in Fig. 2 uses these operators to express a form of liveness property stating that if an input  $i$  is disabled in some state, then there exists a sequence of *tau*-transitions that will enable  $i$ .

```

spec LIVENONDETERMINISTICAUTOMATA =
  sort In
  op tau : In
  sort State
  then free %modal{
    type Set ::= { } | { - } (State) | -- ∪ -- (Set; Set)
    op -- ∪ -- : Set × Set → Set,
    assoc, comm, idem, unit { } }
  then cotype State ::= (next : In → Set)
    • ∀ i : In • [next(i)]false ⇒ ⟨next(tau)*⟩⟨next(i)⟩ true
  end

```

**Fig. 2.** Specification of ‘live’ non-deterministic automata using modalities for structured observations.

The example can be recast in the framework of section 1 as follows. The annotation %**modal** leads to extraction of the finite powerset functor  $\mathcal{P}_\omega$ , and

the cotype leads to a functor

$$T X = \mathcal{P}_\omega(X)^{In}$$

On this functor, a canonical predicate lifting is induced via

$$\text{nat}(2^-, 2^T) \cong 2^{T^2} \ni T\{\top\}.$$

The modal logic for structured observations described above is just the modal logic induced by this predicate lifting.

While this form of modal logic for CoCASL is syntactically rather lightweight, it leads to the need of carrying around distinguished presentations (constructors and equations) of datatype in the signatures (and these presentations need to be preserved by signature morphisms). Moreover, the interaction between basic and structured specifications indicated by the annotation `%modal` is rather implicit and hard to grasp. Most severely, the approach can only handle specific predicate liftings, and hence has only limited expressiveness.

### 3 Functors and Liftings in CoCASL

Motivated by the above considerations, we extend CoCASL by explicit notions of functors and modalities. Both these concepts will, in the extended language, give rise to named components of signatures.

Like in modern higher order functional programming languages such as Haskell [19], functors constitute *type constructors* that enrich the type system generated by the signature. The semantics of functors requires that these type constructors are really the object parts of functors, although the action of the functor on maps is not directly syntactically available as higher order functions are not a basic CoCASL language feature (function types and higher order functions may however be specified by the user; cf. [14]).

A functor is introduced by the keyword **functor**. It must be *defined* as either an initial datatype or a final process type. Thus, the definition takes one of the two standard forms

**functor**  $F(X) = \text{free } \{ \text{type } F(X) ::= \dots \}$

or

**functor**  $F(X) = \text{cfree } \{ \text{cotype } F(X) ::= \dots \}$

where the omitted parts are basic specifications consisting of a single type or cotype declaration, respectively, optionally followed by declarations of additional operation and predicate symbols, as well as

- in the case of a free type declaration, Horn axioms constraining the type as well as the additional operations (typically providing recursive definitions for the latter)

- in the case of a cofree cotype declaration, modal axioms over the cotype  $F(X)$  and corecursive definitions of the additional operations and predicates.

Note that the use of modal axioms in the second case does not constitute a circularity:  $F(X)$  is defined as the final coalgebra for a previously declared functor, for which modalities have already been defined. Additional operations and predicates introduced along with the definition of the functor are uniquely defined due to the freeness or cofreeness constraint, respectively.

The format of functor definitions has been chosen in such a way that, given any interpretation of the functor argument  $X$  as a set  $A$ , the type  $F(X)$  has an interpretation which is determined uniquely up to isomorphism and depends functorially on  $A$ ; thus,  $F$  induces an endofunctor  $\llbracket F \rrbracket$  on the category of sets. In the case of a free datatype, this functor takes a set  $A$  to the interpretation of  $F(X)$  in the initial model of the defining specification interpreting  $X$  as  $A$ , and correspondingly with initial models replaced by fibre-final models in the case of a cofree cotype (cf. [14] for the definition of fibre finality).

At the level of the static semantics, functor definitions such as the above have the effect of extending the signature by a type constructor ( $F$  in the above examples), annotated with the basic specification in curly brackets. The latter, in turn, has an enlarged local environment where the functor argument ( $X$  in the above examples) appears as an additional sort symbol; of course, the functor argument is hidden in the subsequent specification.

In the model semantics, the functor definition as such does not have any effect at all — there is no need for recording an explicit interpretation of the functor in the models, as the interpretation is already determined (up to isomorphism) by the remaining parts of the model. This interpretation shows up, however, as soon as the functor is actually used. As functors are regarded as type constructors, we have a type formation rule producing for every type  $s$  and every functor  $F$  a type  $F(s)$ , whose interpretation is obtained from the interpretation of  $s$  by applying the functor  $\llbracket F \rrbracket$  induced by  $F$  as described above. A typical use of functors is in types of observers for cotypes; in particular, a coalgebra  $X$  for a functor  $F$  is declared by writing

**cotype**  $X ::= (next : F(X))$

The second place where functors may appear is in definitions of predicate liftings. Unary predicate liftings are introduced by means of the keyword **modality** in the form

**modality**  $m : F = \{C \bullet \phi\}$

This declares  $m$  to be a unary predicate lifting for  $F$ , defined as corresponding to the subset  $\{C \mid \phi\}$  of  $F2$  under Prop. 5. Polyadic modalities may be declared in the form

**modality**  $m(\overbrace{--; \dots; --}^{n \text{ placeholders}}) : F = \{C \bullet \phi\}$

The above declares  $m$  to be an  $n$ -ary predicate lifting for  $F$ , corresponding under Prop. 5 to the subset  $\{C \mid \phi\}$  of  $F(2^n)$ .

There is a certain amount of additional syntax available for purposes of defining the properties  $\phi$  above. To begin, the local variable mentioned by  $\phi$  ( $C$  in the above example) need not (and in fact cannot) be provided with a type, being implicitly of type  $F(2^n)$ ; moreover, within the scope of  $\phi$ , further variables without explicit typing may be used (in quantifications) that represent values of type  $2 = \{\perp, \top\}$ . Finally, the elements of  $2$  may be explicitly referred to as terms *true* and *false* (in standard CASL, *true* and *false* are *formulas*). All this serves to encapsulate the mention of  $2$  within definitions of liftings, rather than introducing a type of truth values globally, in an effort to keep the language extension as non-invasive as possible.

CoCASL's original implicit mechanism for defining modalities is kept in the extension: by writing

**modality**  $m : F$  **canonical**

$m$  is defined to be the predicate lifting for  $F$  corresponding to the subset  $F\{\top\}$  of  $F2$ .

The modal operator induced by a predicate lifting  $m$  for  $T$  and an observer operation  $f : X \rightarrow TX$  of a cotype is standardly denoted as  $[m; f]$ . As this is frequently not the desired notation in particular cases, we provide syntax annotations that allow replacing the standard syntax with rather arbitrary notation. Explicitly, the annotation

**modality**  $m(--; \dots; --) : F$  **%syntax**  $[m; --; \dots; --] = C$

replaces the  $n$ -ary modal operator  $[m; --; \dots; --]$  by the mixfix identifier  $C$ , which contains  $n$  placeholders corresponding to the arguments of  $[m; --; \dots; --]$  in the given order.

In this general setting, iterated modal operators  $[m; f^*]$  are defined as greatest fixed points  $[m; f^*]\phi = \nu X. \phi \wedge [m; f]X$ . More precisely, the semantics is defined as the union of all fixed points, which yields a greatest fixed point if the modal operator  $[m; f]$  is monotone. The dual operator is defined by  $\langle m; f^* \rangle \equiv \neg[m; f^*]\neg$ .

**Remark 6.** A basic motivation for introducing functors and modalities as dedicated language features is to avoid the need for extending the language by shallow polymorphism as e.g. in the higher order CASL extension HASCASL [25]. The crucial point here is that polymorphic *axioms* complicate the semantics [24]. It is for this reason that functors and predicate liftings are provided with mandatory definitions, since loosely specified functors or liftings would give rise to implicit polymorphic functoriality or naturality axioms, respectively. Similarly,

defining predicate liftings via the correspondence of Prop. 5 serves the purpose of avoiding polymorphic definitions. Finally, the possibility of including auxiliary operations and predicates in the format for functor definitions provides a comfortable workaround replacing later polymorphic definitions of such entities. If the later introduction of polymorphic operations or predicates is desired by the user, e.g. for purposes of specification structuring, then these may (resp. have to) be emulated by means of parametrized specifications.

**Remark 7.** Like the original version of CoCASL’s modal logic, functors and modal operators induced by predicate liftings may be regarded as a syntactic sugaring of basic CoCASL. Functors may be replaced parametrized specifications of the associated type constructor and the action on morphisms (which has two formal type parameters), and their application with explicit instantiations. In order to code the modal operator  $[m; f]$  induced by a predicate lifting  $m$  for  $T$  and an observer  $f : X \rightarrow TX$  of a cotype  $X$  (the general case of mutually recursive cotypes  $(X_i)$  with observers  $f : X_i \rightarrow TX_j$  and polyadic liftings works analogously), one specifies  $2$  as a free datatype and the predicate type  $2^X$  as a cofree cotype with observer  $is\_in : 2^X \times X \rightarrow 2$ , correspondingly for the predicate type  $2^{T^2}$ . The definition of  $m$  then induces an element of  $2^{T^2}$ .

**Remark 8.** The mechanism for defining functors described above is, as the examples given in the next section will show, quite flexible. It does have its limits, however; in particular, it does not cover definitions of functors  $\mathbf{Set} \rightarrow \mathbf{Set}$  that come about as composites of functors involving a third category, notably composites of the form  $\mathbf{Set} \rightarrow \mathbf{Set}^{\mathbf{op}} \rightarrow \mathbf{Set}$ . This includes e.g. the neighbourhood frame functor and the standard conditional model functor (Examples 4.2 and 4.7). A more general mechanism for functor definitions would include explicit definitions of the action of the functor on morphisms, something we are trying to avoid for reasons given in Remark 6.

## 4 Example Specifications

We now illustrate the specification of modal logics in CoCASL by means of a number of examples, some of them formal specifications of logics given in Example 4.

To begin, Fig. 3 shows a specification of standard modal logic, interpreted over finitely branching Kripke frames. The latter are specified as coalgebras for the finite powerset functor, called  $Set$  in the specification. Recursive functions and predicates needed at later points are defined along with the recursive datatype  $Set(X)$  itself; in particular, the subset predicate is needed for the definition of the modality  $all$  in Fig. 3, and the elementhood predicate is provided for later use in Fig. 5. The modality  $all$  is the box modality of standard modal logic; it arises from the predicate lifting taking a set  $A \subseteq X$  to the set  $Set(A) \subset Set(X)$ , which corresponds according to Prop. 5 to the subset  $\{\emptyset, \{\top\}\}$  of  $Set(2)$ .

The modality  $all$  may alternatively be defined as a canonical modality corresponding to the subset  $Set(\{\top\}) = \{\emptyset, \{\top\}\}$  of  $Set(2)$ :

```

spec FINITEBRANCHING =
functor Set(X) = free {
  type Set(X) ::= {} | {--}(X)
  op -- ∪ -- : Set(X) × Set(X) → Set(X), assoc, comm, idem, unit {}
  preds -- ⊆ -- : Set(X) × Set(X)
          -- ∈ -- : X × Set(X)
  ...      %% recursive definitions of ⊆, ∈
}
modality all : Set = { C • C ⊆ {true} }; %syntax [all; --] = [--]

```

**Fig. 3.** Specification of finitely branching modal logic.

**modality** *all* : *Set* **canonical** %**syntax** [*all*; --] = [--]

It may be used in specifications such as

```

free type Bit ::= 0 | 1
cotype Node ::= (next : Set(Node); out : Bit)
•      out = 0 ⇒ [next] out = 1

```

(with explicit mention of *all* suppressed as prescribed by the syntax annotation) declaring a (loose) transition system with bit-labelled nodes, where all the successors of nodes labelled 0 are labelled 1.

Using cofree cotypes, one can also specify the full class of Kripke frames, without a bound on branching, as the semantics of standard modal logic; the corresponding specification is shown in Fig. 4. Here, the functor *Set* denotes the full powerset functor, specified as a cofree cotype observed via a boolean elementhood function.

```

spec UNBOUNDEDBRANCHING =
free type Bool ::= F | T
functor Set(X) = cofree { cotype Set(X) ::= ( -- ∈ -- : X → Bool ) }
modality all : Set = { C • false ∈ C = F }; %syntax [all; --] = [--]

```

**Fig. 4.** Specification of modal logic with unbounded branching.

As an example of a binary modality, a standard modality for the composite of the finite powerset functor and the squaring functor is specified

in Fig. 5. This functor is well suited for the modelling of processes that may fork into independent subprocesses, e.g. for purposes of higher order communication [27] or mobility [6]; it does not admit an expressive set of unary modalities [22], so that the use of a binary modality cannot in general be avoided. The specification in Fig. 5 imports the definition of the finite powerset functor from Fig. 3; the composition of this functor with the squaring functor  $Pair$  is realized by means of a free datatype  $PairSet(X)$  encapsulating the type  $Set(Pair(X))$ . The binary modality  $bAll$  is determined by the binary predicate lifting taking a pair  $(A, B)$  of subsets of  $X$  to the subset  $\{C \mid (a, b) \in C \implies a \in A \wedge b \in B\}$  of  $PairSet(X)$ , which corresponds under Prop. 5 to the subset  $\{C \mid ((a, b), (c, d)) \in C \implies a = \top \wedge d = \top\}$  of  $PairSet(2^2)$  (where we identify  $2^2$  with  $2 \times 2$ ). The modality  $bAll$  can be used in specifications such as

```

free type Bit ::= 0 | 1
cotype Proc ::= (branch : PairSet(Proc); out : Bit)
• out = 0  $\Rightarrow$  [branch] (out = 1, out = 0)

```

declaring a (loose) cotype  $Proc$  of forking processes with bit-labelled states, where all left (right) children of states labelled 0 are labelled 1 (0).

```

spec FORK = FINITEBRANCHING then
functor Pair(X) = free {type Pair(X) ::= pair(X; X)}
functor PairSet(X) = free {
  type PairSet(X) ::= pairSet(Set(Pair(X)))
  pred _  $\in$  _ : Pair(X)  $\times$  PairSet(X)
  vars z : Pair(X); A : Set(Pair(X))
  • z  $\in$  pairSet(A)  $\Leftrightarrow$  z  $\in$  A
}
modality bAll(--; --) : PairSet = {C •  $\forall a, b, c, d$  •
  pair((a, b), (c, d))  $\in$  C  $\Rightarrow$  a = true  $\wedge$  d = true}
  %syntax [bAll; --] = [--]

```

**Fig. 5.** A binary modality for the ‘forking functor’  $\lambda X. \mathcal{P}(X \times X)$ .

As an illustration of a coalgebraic semantics which is more clearly distinct from standard Kripke semantics, a specification of graded modal logic, interpreted over coalgebras for the bag functor, is shown in Fig. 6. The bag functor is obtained, like the finite powerset functor, from basic operations representing empty, singletons, and union, without however imposing idempotence on the union operator. The modal operators of GML are induced by predicate liftings

$\lambda^k$  as in Example 4.5. The correspondence of Prop. 5 takes  $\lambda^k$  to the subset  $\{n\top + m\perp \mid n > k\}$  of  $Bag(2)$ .

The following example specification of a bag-branching bit-labelled process type expresses that there are always more successors labelled 1 than successors labelled 0:

```

free type Bit ::= 0|1
cotype M ::= (next : Bag(M); out : Bit)
var n : Nat
  •  $\langle n, next \rangle out = 0 \Rightarrow \langle n + 1, next \rangle out = 1$ 

```

Note how quantification over indices of modal operators increases expressivity; indeed, the above formula corresponds to the formula  $M(out = 1)$  of *majority logic* [16], which is not standardly expressible in graded modal logic.

```

spec GRADEDMODALLOGIC =
functor Bag(X) = free {
  type Bag(X) ::= { } | { -- }( X )
  op -- ∪ -- : Bag(X) × Bag(X) → Bag(X),
    assoc, comm, unit { }
  op count : Bag(X) × X → Nat
  ...           %% recursive definition of count
}
var n : Nat
modality more(n) : Bag = { C • count(C, true) > n }
                                     %%syntax [more(n); --] = ⟨n; --⟩

```

**Fig. 6.** Specification of Graded Modal Logic.

We conclude with a few examples illustrating the interpretation of starred modalities:

- Over the cotype *Node* specified above,  $[next*]$  corresponds to the CTL operator  $AG$ , i.e.  $[next*]\phi$  holds in a state if all states reachable from it in finitely many steps satisfy  $\phi$ . The dual  $\langle next* \rangle$  of  $[next*]$  corresponds to the CTL operator  $EF$ .
- If we introduce a separate diamond operator for *Set*

```

modality ex : Set = { C •  $\neg(false \in C)$  }

```

then  $[ex; next]$  corresponds to the CTL operator  $EG$ , with dual  $AF$ .

- Over the bag-branching cotype  $M$  defined above,  $[more(2)*]\phi$  is satisfied in a state  $x : M$  iff  $\phi$  holds everywhere on some bag-branching substructure of  $M$  with root  $x$  in which every node has at least 2 children, counting multiplicities.

## 5 Conclusion

We have proposed a syntactic integration of recent forms of coalgebraic modal logic into CoCASL. The main device is a syntax that makes functors and predicate liftings explicit, replacing less flexible implicit mechanisms in the original CoCASL design. This leads to both a cleaner (static) semantics of CoCASL specifications and to increased expressiveness: in the resulting coalgebraic modal logic, one can now express graded modal logic, majority logic, and probabilistic modal logic, as well as binary modalities. We have illustrated these concepts by means of extensive example specifications.

Once these extensions are incorporated into the CoCASL tool support, coalgebraic modal logic will be embedded into an extensive network of related specification languages and tools, in particular theorem provers, within the Bremen heterogeneous tool set Hets [13]. This will also provide a suitable framework for experimental implementations of generic decision procedures for coalgebraic modal logic [26].

## References

- [1] M. Bidoit and P. D. Mosses. *CASL User Manual*, volume 2900 of *LNCS*. Springer, 2004.
- [2] B. Chellas. *Modal Logic*. Cambridge, 1980.
- [3] C. Cîrstea and D. Pattinson. Modular construction of modal logics. In *Concurrency Theory*, volume 3170 of *LNCS*, pages 258–275. Springer, 2004.
- [4] G. D’Agostino and A. Visser. Finality regained: A coalgebraic study of Scott-sets and multisets. *Arch. Math. Logic*, 41:267–298, 2002.
- [5] H. H. Hansen and C. Kupke. A coalgebraic perspective on monotone modal logic. In J. Adámek and S. Milius, editors, *Coalgebraic Methods in Computer Science*, volume 106 of *ENTCS*, pages 121–143. Elsevier, 2004.
- [6] D. Hausmann, T. Mossakowski, and L. Schröder. A coalgebraic approach to the semantics of the ambient calculus. *Theoret. Comput. Sci.* to appear. Preliminary version appeared in J. Fiadeiro, N. Harman, M. Roggenbach, and J. Rutten (eds.), *Algebra and Coalgebra in Computer Science*, vol. 3629 of *LNCS*, Springer, 2005, pp. 232–246.
- [7] A. Heifetz and P. Mongin. Probabilistic logic for type spaces. *Games and Economic Behavior*, 35:31–53, 2001.
- [8] B. Jacobs. Towards a duality result in the modal logic of coalgebras. In *Coalgebraic Methods in Computer Science*, volume 33 of *ENTCS*. Elsevier, 2000.
- [9] C. Kupke, A. Kurz, and D. Pattinson. Ultrafilter extensions for coalgebras. In *Algebra and Coalgebra in Computer Science*, volume 3629 of *LNCS*, pages 263–277. Springer, 2005.

- [10] A. Kurz. Specifying coalgebras with modal logic. *Theoret. Comput. Sci.*, 260:119–138, 2001.
- [11] A. Kurz. Logics admitting final semantics. In *Foundations of Software Science and Computation Structures*, volume 2303 of *LNCS*, pages 238–249. Springer, 2002.
- [12] K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inform. Comput.*, 94:1–28, 1991.
- [13] T. Mossakowski. Heterogeneous specification and the heterogeneous tool set. Habilitation thesis, University of Bremen, 2004.
- [14] T. Mossakowski, L. Schröder, M. Roggenbach, and H. Reichel. Algebraic-coalgebraic specification in CoCASL. *J. Logic Algebraic Programming*, 67:146–197, 2006.
- [15] P. D. Mosses, editor. *CASL Reference Manual*, volume 2960 of *LNCS*. Springer, 2004.
- [16] E. Pacuit and S. Salame. Majority logic. In *Principles of Knowledge Representation and Reasoning*, pages 598–604. AAAI Press, 2004.
- [17] D. Pattinson. Semantical principles in the modal logic of coalgebras. In *Symposium on Theoretical Aspects of Computer Science*, volume 2010 of *LNCS*, pages 514–526. Springer, 2001.
- [18] D. Pattinson. Expressive logics for coalgebras via terminal sequence induction. *Notre Dame J. Formal Logic*, 45:19–33, 2004.
- [19] S. Peyton-Jones, editor. *Haskell 98 Language and Libraries — The Revised Report*. Cambridge, 2003. also: *J. Funct. Programming* **13** (2003).
- [20] M. Röfßiger. Coalgebras and modal logic. In *Coalgebraic Methods in Computer Science*, volume 33 of *ENTCS*. Elsevier, 2000.
- [21] J. Rutten. Universal coalgebra: A theory of systems. *Theoret. Comput. Sci.*, 249:3–80, 2000.
- [22] L. Schröder. Expressivity of coalgebraic modal logic: the limits and beyond. In *Foundations of Software Science And Computation Structures*, volume 3441 of *LNCS*, pages 440–454. Springer, 2005. Extended version to appear in *Theoret. Comput. Sci.*
- [23] L. Schröder. A finite model construction for coalgebraic modal logic. In L. Aceto and A. Ingólfssdóttir, editors, *Foundations Of Software Science And Computation Structures*, volume 3921 of *LNCS*, pages 157–171. Springer, 2006.
- [24] L. Schröder, T. Mossakowski, and C. Lüth. Type class polymorphism in an institutional framework. In J. Fiadeiro, editor, *Recent Developments in Algebraic Development Techniques, 17th International Workshop, WADT 04*, LNCS. Springer, 2004. to appear.
- [25] L. Schröder, T. Mossakowski, and C. Maeder. HASCASL – Integrated functional specification and programming. Language summary. available at <http://www.informatik.uni-bremen.de/agbkb/forschung/formal.methods/CoFI/HasCASL>, 2003.
- [26] L. Schröder and D. Pattinson. PSPACE bounds for rank 1 modal logics, 2006.
- [27] B. Thomsen. A theory of higher order communicating systems. *Inform. and Comput.*, 116:38–57, 1995.