

Algebraic-coalgebraic specification in CoCASL

Till Mossakowski¹, Horst Reichel², Markus Roggenbach¹, and Lutz Schröder¹

¹ BISS, Dept. of Computer Science, University of Bremen

² Institute for Theoretical Computer Science, Technical University of Dresden

Abstract. We introduce CoCASL as a simple coalgebraic extension of the algebraic specification language CASL. CoCASL allows the nested combination of algebraic datatypes and coalgebraic process types. We show that the well-known coalgebraic modal logic can be expressed in CoCASL. We present sufficient criteria for the existence of cofree models, also for several variants of nested cofree and free specifications. Moreover, we describe an extension of the existing proof support for CASL (in the shape of an encoding into higher-order logic) to CoCASL.

In recent years, coalgebra has emerged as a convenient and suitably general way of modeling the reactive behaviour of systems [26]. While algebraic specification deals with *inductive datatypes* generated by constructors, coalgebraic specification deals with *coinductive process types* that are observable by selectors. An important role is played here by *final coalgebras*, which are complete sets of possibly infinite behaviours, such as streams or even the real numbers.

For algebraic specification, the Common Algebraic Specification Language CASL [20] has been designed as a unifying standard, while for the much younger field of coalgebraic specification there is still a divergence of notions and notations. The idea pursued here is to obtain a fruitful synergy by extending CASL with coalgebraic constructs that dualize (in the sense of e.g. [5]) the algebraic constructs already present in CASL.

In more detail, CoCASL provides a basic co-type construct, cogeneratedness constraints, and structured cofree specifications; moreover, coalgebraic modal logic is introduced as syntactical sugar. Co-types serve to describe reactive processes, equipped with observer operations whose role is dual to that of the constructors of a datatype. Cotypes can be qualified as being cogenerated or cofree, respectively, thus imposing full abstractness and realization of all observable behaviours, respectively. The most powerful construct are cofree specifications, which allow specifying final models of arbitrary specifications. Of course, this raises the question for what kinds of specifications such final models actually exist. We provide a sufficient existence condition which covers specifications that employ initially specified datatypes in observer functions and restrict behaviours by modal formulas. For such cases, we also lay out how the existing proof support for CASL, realized by means of an encoding into Isabelle/HOL [16], can be extended to CoCASL. In summary, CoCASL is a syntactically and semantically simple extension of CASL that allows a straightforward treatment of reactive behaviour.

1 CASL

The specification language CASL (*Common Algebraic Specification Language*) has been designed by COFI, the international *Common Framework Initiative for Algebraic Specification and Development* [20]. Its features include first-order logic, partial functions, subsorts, sort generation constraints, and structured and architectural specifications. For the language definition and a full formal semantics cf. [20]. An important point here is that the semantics of structured and architectural specifications is independent of the logic employed for basic specifications, so that the language is easily adapted to the extension of the logic envisaged here. That said, CoCASL does introduce one additional structuring concept, namely, cofree specifications.

We now briefly recall the many-sorted CASL institution; subsorting is then defined on top of this. Full details can be found in [20, 17]. A CASL signature consists of a set of sorts, two sets of (total and partial) function symbols, and a set of predicate symbols (each symbol coming with a string of argument sorts and, for function symbols, a result sort). Signature morphisms map the four components in a compatible way. Models are many-sorted partial first order structures. Homomorphisms are so-called weak homomorphisms. That is, they are total as functions, and they preserve (but not necessarily reflect) the definedness of partial functions and the satisfaction of predicates.

Over such a signature, sentences are built from atomic sentence using the usual features of first order logic. Here, an atomic sentence is either a definedness assertion, a strong equation, an existence equation, or a predicate application; see [20] for details. There is an additional type of sentence that goes beyond first-order logic: a *sort generation constraint* states that a given set of sorts is generated by a given set of functions, i.e. that all the values of the generated sorts are reachable by some term in the function symbols, possibly containing variables of other sorts.

The CASL language is defined on top of this institution, offering a richer and more convenient syntax than the plain CASL institution. For instance, it provides powerful constructs for defining datatypes, which are briefly recalled below, in direct comparison to the corresponding CoCASL constructs.

2 Type and cotype definitions

The basic CASL construct for type definitions is the **types** construct. It declares constructors and, optionally, selectors for a datatype (or several datatypes at once); both constructors and selectors may be partial. Such a type declaration is expanded into the declaration of the constructor and selector operations and axioms relating the selectors and constructors. Nothing else is said about the type; thus, there may not only be ‘junk’ and ‘confusion’, but there may also be rather arbitrary behaviour of the selectors outside the range of the corresponding constructors.

In CoCASL, this construct is complemented by the **cotypes** construct. In its full form, the syntax of this construct is identical to the **types** construct; e.g., one may write

cotype $Process ::= cont(hd1 :?Elem; next :?Process)$
 $| fork(hd2 :?Elem; left :?Process; right :?Process)$

thus determining constructors and selectors as for types. However, for cotypes, the constructors are optional and the selectors are mandatory. Moreover, the **cotype** construct introduces a number of additional axioms concerning the domains of definition of the selectors, besides the axioms relating constructors with their selectors as for types:

- the domains of two selectors in the same alternative are the same,
- the domains of two selectors in different alternatives are disjoint, and
- the domains of all selectors of a given sort are jointly exhaustive.

Thus, the alternatives in a cotype are to be understood as parts of a disjoint sum.

Definition 1. A cotype in CoCASL is given by the local environment sorts and the family of observers

$$CT = (S, (obs_{i,j,k} : T_i \rightarrow T_{i,j,k})_{i=1..n, j=1..m_i, k=1..r_{i,j}}).$$

Here, S is a set of sorts (the local environment sorts, also called observable sorts), $T_1 \dots T_n$ are the newly declared process types (or non-observable sorts) in the cotype (which possibly involve mutual recursion, and $obs_{i,j,k}$ is the k -th observer of the j -th alternative in the **cotype** definition of T_i . $T_{i,j,k}$ is the result sort of the observer; it may be either one of the T_i or one of the local environment sorts in S . The *signature* $Sig(CT)$ of a cotype CT consists of the local environment sorts S , the cotype sorts $T_1 \dots T_n$, and the profiles of the observers; the *theory* $Th(CT)$ also adds the above listed axioms.

Cotypes correspond directly to coalgebras:

Proposition 1. To a given CoCASL **cotype** definition CT , one can associate a functor $F : \mathbf{Set}^n \rightarrow \mathbf{Set}^n$ such that the category of partial $Th(CT)$ -algebras is isomorphic to the category of F -coalgebras. In particular, this implies that all homomorphisms between partial $Th(CT)$ -algebras are closed [6], i.e. not only preserve, but also reflect definedness.

3 Generation and cogeneration constraints

In order to exclude ‘junk’ from models of datatypes, CASL provides generat- edness constraints that essentially introduce (higher order) implicit induction axioms. Dually, CoCASL introduces *cogeneratedness constraints* that amount to an implicit coinduction axiom and thus restrict the models of the datatype to fully abstract ones. This means that equality is the largest congruence w.r.t. the

```

spec STREAM1 [sort Elem] =
  cogenerated cotype
    Stream ::= cons(hd : Elem; tl : Stream)
end

```

Fig. 1. Cogenerated specification of bit streams in CoCASL

introduced sorts, operations and predicates (excluding the constructors). In the example in Fig. 1, the STREAM-models are (up to isomorphism) the *tl*-closed subsets of E^ω , where E is the interpretation of the sort *Elem*. A more complex example is the specification of CCS [18]. States are generated by the CCS syntax, but they are identified if they are bisimilar w.r.t. the ternary transition relation. This can be expressed in CoCASL by stating that states are cogenerated w.r.t. the transition relation.

Formally, a *cogeneration constraint* over a signature Σ is a subsignature fragment (i.e. a tuple of component-wise subsets that need not by itself form a complete signature) $\bar{\Sigma} = (\bar{S}, \bar{TF}, \bar{PF}, \bar{P})$ of Σ . In the above example, the cogeneration constraint is $(\{Elem\}, \{hd, tl\}, \emptyset, \emptyset)$.

A Σ -cogeneration constraint $\bar{\Sigma} \subseteq \Sigma$ is satisfied in a Σ -model M if each equivalence relation on M that

- is the equality relation on sorts in $\bar{\Sigma}$, and
- is a closed congruence for the operations and predicates in $\bar{\Sigma}$

is the equality relation. (Recall from [6] that a congruence on a partial algebra is closed if domains of partial functions and predicates are closed under the congruence.)

Note that selectors of cotypes, which play the role of observers, are always unary. However, like the **generated** $\{ \dots \}$ construct in CASL, the **cogenerated** $\{ \dots \}$ construct allows the inclusion of arbitrary signature items in the cogeneratedness constraint, so that observers of arbitrary arity are also possible. In particular, observers may have additional observable arguments (cf. the example in Fig. 6 below) as well as several non-observable arguments.

In duality to generated types in CASL, the construct **cogenerated cotype** \dots abbreviates **cogenerated** $\{ \text{cotype } \dots \}$. No such abbreviation is provided for **cogenerated** $\{ \text{type } \dots \}$, the use of which is in fact expressly discouraged. A particularly discouraging example for the use of types where cotypes are expected is given in 6.

4 Free types and cofree cotypes

CASL allows the exclusion not only of ‘junk’ in datatypes, but also of ‘confusion’, i.e. of equalities between different constructor terms. To this end, it provides the

(basic) **free types** construct. Free datatypes carry implicit axioms that state, beside term-generatedness, the injectivity of the constructors and the disjointness of their images. The most immediate effect of these axioms is that *recursive definitions on a free datatype are conservative*. The elements of a free datatype can be thought of as being the (finite) constructor terms, i.e. in a suitable sense finite trees.

```

spec STREAM2 [sort Elem] =
  cofree cotype
    Stream ::= (hd : Elem; tl : Stream)
end

```

Fig. 2. Cofree specification of bit streams in CoCASL.

In CoCASL, we provide, dually, a **cofree cotypes** construct that specifies the absolutely final coalgebra of infinite behaviour trees (see Example 6 on why there is no **cofree types** construct). More concretely, this means that, in addition to cogeneratedness, there is also a principle stating that there are enough behaviours, namely all infinite trees [2] (with branching as specified by the selectors). In contrast to its dual (no confusion among constructors), the latter principle cannot be expressed in first-order logic; however, a second-order specification is possible (see below). In the example in Fig. 2, the STREAM2-models are isomorphic to E^ω , where E is the interpretation of the sort *Elem*.

Definition 2. Given a set of sorts S , an S -colouring is just an S -sorted family of sets (of colours).

We are now ready to dualize the important algebraic concept of term algebra.

Definition 3. Given a cotype

$$CT = (S, (obs_{i,j,k} : T_i \rightarrow T_{i,j,k})_{i=1\dots n, j=1\dots m_i, k=1\dots r_{i,j}})$$

and an S -colouring C , the *behaviour algebra* $Beh_{CT}(C)$ is defined to be the following $Sig(CT)$ -algebra:

- the carriers for observable sorts (i.e. in S) are those determined by C ;
- the carriers for a non-observable sort T_{i_0} consist of all infinite trees of the following form:
 - each inner node is labelled with a pair (T_i, j) , where T_i is a non-observable sort and $j \in \{1 \dots m_i\}$ selects an alternative out of those for T_i ;
 - the root is labelled with (T_{i_0}, j_0) for some j_0 ;
 - each leaf is labelled with an observable sort $s \in S$ and some colour from C_s ;

- each non-leaf node with label (T_i, j) has one child for each of the observers $obs_{i,j,k}$ ($k = 1 \dots r_{i,j}$). The child node is labelled with the result sort of the observer.
- an observer operation $obs_{i_0,j,k}$ is defined for a tree with root (T_{i_0}, j_0) if and only if $j = j_0$, and in this case, it just selects the child tree corresponding to the observer.

Proposition 4. *Given a cotype*

$$CT = (S, (obs_{i,j,k} : T_i \rightarrow T_{i,j,k})_{i=1\dots n, j=1\dots m_i, k=1\dots r_{i,j}})$$

and an S -colouring C , the behaviour algebra $Beh_{CT}(C)$ is final in the following sense: for any $Sig(CT)$ -algebra A equipped with a C -colouring h (that is, a family of maps $(h_s : A_s \rightarrow C_s)_{s \in S}$), we can extend h in a unique way to a $Sig(CT)$ -homomorphism

$$h^\natural : A \rightarrow Beh_{CT}(C).$$

Proof. Using the characterization of Prop. 1, the result follows from the general construction of final coalgebras over the category of $\{T_1, \dots, T_n\}$ -sorted sets (this generalizes the well-known result for **Set** [2]). Intuitively, h^\natural constructs the behaviour of an element, which is the infinite tree given by all possible observations that can be made successively applying the observers until a value of observable sort (i.e. in S) is reached. \square

Given a signature Σ , we formally add cofreeness constraints of form $cofree(CT)$, where

$$CT = (S, (obs_{i,j,k} : T_i \rightarrow T_{i,j,k})_{i=1\dots n, j=1\dots m_i, k=1\dots r_{i,j}})$$

is a cotype with $Sig(CT) \subseteq \Sigma$, as Σ -sentences to our logic. A cofreeness constraint $cofree(CT)$ holds in a Σ -algebra A , if the reduct of A to $Sig(CT)$ is isomorphic to the behaviour algebra $Beh_{CT}(C)$ over the set of colours C with $C_s := A_s$ for $s \in S$.

Note that this implies the satisfaction of the cogeneratedness constraint $(S, \{sel_{i,j,k} | sel_{i,j,k} \text{ total}\}, \{sel_{i,j,k} | sel_{i,j,k} \text{ partial}\}, \emptyset)$, i.e. each cofree cotype is also cogenerated. The converse does not hold, i.e. a cogenerated cotype need not be cofree. However, cogenerated *cotypes* still behave quite nicely (in contrast to arbitrary cogenerated *types*): the elements of carriers of the non-observable sorts (i.e. those outside S) are completely determined by their *behaviours*. Thus, the elements can be identified with their behaviours, and up to isomorphism, we have a subalgebra of the cofree cotype. Hence, cofreeness essentially adds the requirement that each possible behaviour is actually represented by an element.

Note that an equivalent description of the behaviour algebra can be given in terms of contexts [11], using the “magic formula”:

$$(\Pi_{v \in \bar{S}} [Ctx_{(\bar{S}, (\bar{F}_s)_{s \in S \setminus \bar{S}})}^\Sigma](C)[z_s]_v \rightarrow C_v])_{s \notin \bar{S}}$$

where $Ctx_{(\bar{s}, (\bar{F}_s)_{s \in S \setminus \bar{s}})}^\Sigma(C)[z_s]$ is the set of all terms consisting of constants in C , observer operations and a single occurrence of a variable z_s of non-observable sort s .

The main benefit of cofree cotypes (in comparison to cogenerated cotypes) is the principle

corecursive definitions in cofree cotypes are always conservative

meaning that any given model of a cofree cotype can (even uniquely) be extended to a model of a corecursive definition over this cotype.

Note that if we want to get an institution, in order to be able to translate the various constraints along signature morphisms in a way that the satisfaction condition for institutions is fulfilled, one has to equip the constraints with an additional signature morphism, as in [20, 17].

Fig. 3 contains a summary of the CASL concepts and their CoCASL dualizations (including structured free and cofree covered in the next section).

Algebra	Coalgebra
type = (partial) algebra	cotype = coalgebra
constructor	selector
generation	observability
generated type	cogenerated (co)type
= no junk	= full abstractness
= induction principle	= coinduction principle
no confusion	all possible behaviours
free type	cofree cotype
= absolutely initial datatype	= absolutely final process type
= no junk + no confusion	= full abstractness + all possible behaviours
free { ... } = initial datatype	cofree { ... } = final process type

Fig. 3. Summary of dualities between CASL and CoCASL.

5 Structured free and cofree specifications

Besides institution-specific language constructs, CASL also provides institution-independent structuring constructs. In particular, CASL provides the structured **free** construct that restricts the model class to *initial* or *free* models. That is, if Sp_1 is a specification with signature Σ_1 , then the models of Sp_1 **then free** $\{Sp_2\}$ are those models M of Sp_1 **then** Sp_2 that are free over $M|_{\Sigma_1}$ w.r.t. the reduct functor $-|_{\Sigma_1}$. This allows for the specification of datatypes that are generated freely w.r.t. given axioms, as, for example, in the specification of finite sets over an element sort shown in Figure 4.

The **cofree** $\{ \dots \}$ construct dualizes the **free** $\{ \dots \}$ construct by restricting the model class of a specification to the cofree ones. This generalizes **cofree**

```

spec GENERATEFINITESET [sort Elem] =
free
  { type FinSet[Elem] ::= { }
      | { -- }( Elem )
      | -- ∪ -- ( FinSet[Elem]; FinSet[Elem] )
      op -- ∪ -- : FinSet[Elem] × FinSet[Elem] → FinSet[Elem],
          assoc, comm, idem, unit { }
  }
end

```

Fig. 4. Specification of finite sets over an arbitrary element sort in CASL.

types to the case of non-unary functions (e.g. as in Figure 5) and the presence of axioms (e.g. as in Figure 7 below).

More precisely, the semantics of **cofree** is defined as follows:

Definition 5. If Sp_1 is a specification with signature Σ_1 then the models of Sp_1 **then cofree** $\{Sp_2\}$ are those models M of Sp_1 **then** Sp_2 that are *fibre-final* over $M|_{\Sigma_1}$ w.r.t. the reduct functor $_ |_{\Sigma_1}$. Here, fibre-finality means that M is the final object in the fibre over $M|_{\Sigma_1}$. The fibre over $M|_{\Sigma_1}$ is the full subcategory of $Mod(Sp_2)$ consisting of those models whose Σ_1 -reduct is $M|_{\Sigma_1}$.

This definition deviates somewhat from the semantics of **free** in that the latter postulates *freeness* rather than fibre-initiality. (Actually, it might be worthwhile to redefine the CASL semantics for free specifications in terms of fibre-initial models.) We will see shortly that the more liberal semantics for **cofree** is essential in cases where sorts from the local environment occur as argument sorts of selectors. Call a sort from the local environment an *output sort* if it occurs only as a result type of selectors. In the cases of interest, a more general co-universal property concerning, in the notation of the above definition, morphisms of Σ_1 -models that are the identity on all sorts except possibly the output sorts follows from fibre-finality.

We shall see below (Theorem 11) that the **cofree cotypes** construct is equivalent to **cofree { cotypes ... }**. By contrast, the use of **cofree { types ... }** should be avoided:

Example 6. The specification

```

free type Bool ::= false | true then
cofree { type T ::= c1(s1 :?Bool) | c2(s2 :?Bool) },

```

is inconsistent. Indeed, by applying the uniqueness part of finality to a model of the unrestricted type where T has an element on which both selectors are undefined (this is allowed for types but not for cotypes), one obtains that any model of the cofree type would be a singleton; however, singleton models fail to satisfy the finality property e.g. for the model of the unrestricted type where T is $Bool \times Bool$ and the selectors are the projections.

```

spec FUNCTIONTYPE =
  sorts A, B
  then cofree {
    sort Fun[A, B]
    op eval : Fun[A, B] × A → B
  }
end

```

Fig. 5. Cofree specification of function types.

As an example for the significance of the relaxation of the cofreeness condition, consider the specification of Moore automata as given in Figure 6. Here, the observer *next* depends not only on the state, but additionally on an input letter.

```

spec MOORE =
  sorts In, Out
  then cofree {
    sort State
    ops next : In × State → State
        observe : State → Out
  }
end

```

Fig. 6. Cofree specification of Moore automata.

In the standard theory of coalgebra, *next* would become a higher-order operation $next : State \rightarrow State^{In}$, and the cofree coalgebra indeed yields the final automaton showing all possible behaviours - but only for a *fixed* carrier for *In* (the inputs). The carrier for *Out* is also regarded as fixed; however, one can show that the co-universal property holds also for morphisms that act non-trivially on *Out*. If the semantics of **cofree** required actual cofreeness, i.e. a couniversal property also for morphisms that act non-trivially on *In*, the specification would be inconsistent!

Let us now come to a further modification of the stream example. If the axiom were omitted in the specification in Figure 7, the model class would be the same as that in Figure 1, instantiated to the case of bits as elements. *With* the axiom, the streams are restricted to those where two 0's are always followed by a 1. Again, this is unique up to isomorphism.

It is straightforward to specify iterated free/cofree constructions, similarly as in [23]. Consider e.g. the specification of lists of streams of trees in Figure 8. Alternatively, one could have used structured free and cofree constructs as well:

```

spec BITSTREAM3 =
  free type Bit ::= 0 | 1
  then cofree {
    cotype BitStream ::= (hd : Bit; tl : BitStream)
     $\forall s : \textit{BitStream}$ 
    •  $hd(s) = 0 \wedge hd(tl(s)) = 0 \Rightarrow hd(tl(tl(s))) = 1$  }
  end

```

Fig. 7. Structured cofree specification of bit streams in COCASL.

SP then free {*SP*₁} **then cofree** {*SP*₂} **then free** ...

Note that also in the latter case, there won't be any **free** *within* a **cofree** or vice versa.

```

spec LISTSTREAMTREE [sort Elem] =
  free type
    Tree ::= EmptyTree | Tree(left :? Tree; elem :? Elem; : Elem; right :? Tree)
  cofree cotype
    Stream ::= (hd : Tree; tl : Stream)
  free type
    List ::= Nil | Cons(head :? Stream; tail :? List)
  end

```

Fig. 8. Nested free and cofree (co)types.

An example for **free** *within* **cofree** is shown in Figure 9. Here, the inner **free** has to be a structured one, since sets cannot be specified as **free type** directly. Alternatively, sets may be specified using a **generated type** together with a first-order extensionality axiom. We have preferred the former variant over the latter one in order to be able to apply Theorem 10 below.

6 Modal logic

Given a cotype that defines non-observable sorts with selectors acting as observers, we can define a coalgebraic modal logic in the style of [14]. A non-observable sort corresponds to a set of possible worlds, and leaving this set implicit is the main idea of modal logic. A selector *t* with *non-observable* result sort leads to modalities [*t*], <*t*>, [*t**], <*t**> (all-next, some-next, always, eventually), while a selector with observable result sort leads to a flexible constant (i.e. a constant that depends on the respective world) that can be used to make observations (by equating it to a term of observable sort). The modal logic then

```

spec NONDETERMINISTICAUTOMATA =
  sort In
  then cofree {
    sort State
    then free {
      type Set ::= {} | {--}(State) | -- ∪ --(Set; Set)
      op -- ∪ -- : Set × Set → Set,
          assoc, comm, idem, unit { } }
    then op next : In × State → Set }
  end

```

Fig. 9. A free type *within* a cofree type.

consists of such equations as atomic sentences, which may be combined using the modalities above and the usual propositional connectives, as well as quantification over variables of observable sorts. Using this logic, we can write, in the example of Figure 7,

$$hd = 0 \wedge \langle tl \rangle hd = 0 \Rightarrow \langle tl \rangle \langle tl \rangle hd = 1$$

as syntactic sugar for

$$hd(s) = 0 \wedge hd(tl(s)) = 0 \Rightarrow hd(tl(tl(s))) = 1$$

Each modal formula ϕ has a *sort*, and is implicitly universally quantified over a variable of this sort. The sort of ϕ is determined by the non-observable argument in observers used in ϕ . In particular, a modal formula is well-formed only in case of correct sorting. One may switch to a different sort (i.e. a different state space) using the modalities, but only in a well-sorted way. If necessary (due to overloading), observers have to be provided with explicit types. The ‘iterative’ modalities $[t^*]$ and $\langle t^* \rangle$ are meaningful only for observers that remain within the same non-observable sort.

Moreover, we provide a *global diamond* $\langle \text{global} \rangle$ as suggested in [15], where $\langle \text{global} \rangle \phi$ expresses the fact that ϕ holds in some state of the system. For reasons laid out in [15], the global diamond is restricted to positive occurrences. As explained in [15], the global diamond is, in terms of expressivity, equivalent to *modal rules* which state implications between *validities* of modal formulas. For full details of the modal logic see [19].

The modal logic allows expressing safety or fairness properties. For example, the model of the specification BITSTREAM4 of Figure 10 consists, up to isomorphism, of those bitstreams that will always eventually output a 1. Here, the ‘always’ stems from the fact that the modal formula is, on the outside, implicitly quantified over all states, i.e. over all elements of type *BitStream*.

Remark 7. The modal μ -calculus [13], which provides a syntax for least and greatest fixed points of recursive modal predicate definitions, is expressible using free and cofree specifications: μ is expressible with free recursively defined

```

spec BITSTREAM4 =
  free type Bit ::= 0 | 1
  then cofree {
    type BitStream ::= (hd : Bit; tl : BitStream)
    • <tl*> hd = 1
  }
end

```

Fig. 10. Specification of a fairness property.

predicates, while ν is expressible with cofree recursively defined predicates, and nesting of μ and ν corresponds to nesting of **free** and **cofree**. It is an open point of discussion whether future versions of CoCASL should include the μ -calculus, or whether the existing modal operators and the explicit coding of μ -formulas suffice for practical purposes.

7 Existence of cofree models

The theory of algebraic specification and institutions provides us with a very general characterization of the existence of free models [27]: free models exist for specifications with universally quantified Horn axioms. (Part of) a dual result has been obtained in [15]. In summary, results from [26, 14, 15] guarantee that cofree coalgebras over **Set** exist for bounded functors and modal axioms or, more generally, axioms that are stable under coproducts and quotients. This has to be generalized slightly in order to cope with specifications with several non-observable sorts, i.e. for coalgebras over **Set**^{*n*}.

Even more generally, we have

Proposition 8. *Let \mathbf{C} be a category equipped with a factorization system $(\mathbf{E}, \mathcal{M})$ for (large) sinks [1], and let $\Sigma : \mathbf{C} \rightarrow \mathbf{C}$ be a functor that preserves \mathcal{M} . Then $(\mathbf{E}, \mathcal{M})$ lifts to a factorization structure to the category $\mathbf{CoAlg}(\Sigma)$ of Σ -coalgebras.*

Let \mathbf{B} be a subcategory of $\mathbf{CoAlg}(\Sigma)$ that is closed under \mathbf{E} -sinks, and let $\mathbf{CoAlg}(\Sigma)$ have a final coalgebra. Then \mathbf{B} has a fully abstract final coalgebra in the sense of [15].

Proof. The lifting statement is clear. The given condition on \mathbf{B} is equivalent to \mathbf{B} being \mathcal{M} -coreflective [1]. Then the \mathcal{M} -coreflection of the final Σ -coalgebra is a (fully abstract) final coalgebra in \mathbf{B} .

The condition on Σ is always almost satisfied for the factorization structure (jointly surjective, injective) on **Set**^{*n*}, since injective maps in **Set**^{*n*} are sections provided that their domain is non-empty; in fact, by a construction described for $n = 1$ in [3], we can always assume preservation of injective maps.

If \mathbf{C} , and hence $\mathbf{CoAlg}(\Sigma)$, has coproducts, then closedness under \mathbf{E} -sinks is equivalent to closedness under quotients and coproducts, provided that every \mathbf{E} -sink contains a small \mathbf{E} -sink (this is the case in \mathbf{Set}^n). However, it is often just as easy to argue directly via \mathbf{E} -sinks, e.g. in the proof of the following rather general sufficient condition for \mathcal{M} -coreflexivity (and, hence, existence of final coalgebras):

Lemma 9. *Let $\Sigma : \mathbf{Set}^n \rightarrow \mathbf{Set}^n$. If \mathbf{B} is a subcategory of $\mathbf{CoAlg}(\Sigma)$ determined by formulas of the form*

$$\phi \equiv \forall s : S \bullet P(s),$$

where S is one of the carriers of the coalgebra, and

$$P(s) \implies P(h_S(s))$$

for each coalgebra homomorphism h with S -component h_S , then \mathbf{B} is closed under jointly surjective sinks in $\mathbf{CoAlg}(\Sigma)$.

It is easy to see that this condition is satisfied for the modal logic formulas described above.

We are now ready to state the main existence result:

Theorem 10. Let Sp be the specification

$$Sp_1 \text{ then cofree } \{ Sp_2 \}.$$

Call the sorts from Sp_1 *observable* sorts. Let the specification Sp_2 consist of (no more than)

- declarations of *non-observable* sorts
- *auxiliary* datatypes that are *defined* over the other sorts using **free** with only equational axioms (or an equivalent construction)
- further operations called *observers* that each have exactly one non-observable argument and otherwise only observable arguments called *parameters*, and
- modal logic formulas, and
- (mandatory) further axioms (say, a redeclaration of the non-observable sorts as **cotypes**) stating that domains of observers do not depend on parameters and form a disjoint cover of the respective non-observable sorts.

Then Sp is conservative (model-extensive) over Sp_1 .

Proof. From Proposition 1 together with monomorphicity of the auxiliary sorts, we know that the category of Sp_2 -models over a given Sp_1 -model is equivalent to a subcategory of Σ -coalgebras for some functor $\Sigma : \mathbf{Set}^n \rightarrow \mathbf{Set}^n$. It is easy to check that Σ is bounded, and hence admits a final coalgebra [26]; the proof is finished by appealing to Lemma 9 and Proposition 8.

Moreover, we have

Theorem 11. If DD is a sequence of selector-based datatype definitions without subsorting, then

$$\text{cofree } \{ \text{cotypes } DD \} \quad \text{and} \quad \text{cofree cotypes } DD$$

have the same semantics.

8 Tool support

It is quite straightforward to extend the central analysis tool for CASL (the CASL tool set), to CoCASL. Of course, the more challenging task is to obtain good proof support. Here, we aim at extending the existing encoding of CASL in Isabelle/HOL [16] to CoCASL.

cogenerated types are no problem: coinduction is a second-order principle. The same holds for the infinite trees needed for **cofree cotypes**; actually, Isabelle/HOL already comes with such trees.

The greater difficulties come with specifications of the form **cofree** $\{SP\}$. If SP is flattenable and axiomatized within the modal logic, we can proceed similarly to the case of **cofree types**: the model of **cofree** SP is a subcoalgebra of the coalgebra of *all* behaviours (as specified by the corresponding **cofree type**), namely the largest subcoalgebra that satisfies the modal axioms.

More complex examples, such as nondeterministic automata or trees with unbounded branching, involve a free specification of output sorts of selectors (like lists or sets) within a **cofree** $\{\dots\}$. Here, in a first step, we proceed as above and encode the cofree type over the absolutely free type (only the branching may now be infinite, being determined by a datatype). Then the cofree type over the relatively free type is obtained as the quotient modulo the largest congruence [12]. In terms of tool support, this means the following:

- Equality of elements in the cofree datatype is obtained as before by coinductive reasoning (or via terminal sequence induction [21]), the difference with the absolutely free case being that the formulas in the free specification (e.g. associativity, commutativity, and idempotence in the case of finite sets) are now available for such proofs.
- Distinctness of elements is shown, again as before, by establishing that the behaviours are different. Here, the encoding of free specifications comes in: distinctness of two elements of a relatively free type is shown by separating the two elements by a congruence.

Remark 12. Above, we have seen two cases where free specifications within cofree specifications allow good technical handling:

- the output sorts of selectors for a cofree datatype may be given by a free specification, which is handled as described above;
- the modal formulas that restrict the elements of the cofree type may involve freely (or cofreely) specified predicates, which are dealt with in Isabelle by means of least and greatest fixed points.

Beyond these two cases, the situation remains somewhat unclear. E.g., the following specification is inconsistent:

```
spec FINALELEMENT = BOOL then
cofree {
  free type Unit ::= 1
  op el : Unit → Bool
}
```

This seems to indicate that input sorts should not be restricted by equational axioms (the freeness constraint can be replaced by an equation here), or in fact by anything else except modal formulas; this is in agreement with suggestions made in [14]. On the other hand, observe that constraining output sorts is more or less mandatory: e.g., the specification of Moore automata (Figure 6) becomes meaningless if the freeness constraint (for the type of sets) is omitted — the model it describes is then just the singleton set (with a ‘power set’ consisting only of the empty set). In other words, enough of a handle must be provided to actually prove distinctness of observations.

Also, a proof principle for free specifications containing cofree specifications seems to be much harder to obtain. Here, we propose to avoid the outer free specification and use a generation axiom plus some characterization of equality by suitably chosen observers instead.

9 Conclusion and related work

We have introduced CoCASL as a relatively easy extension of CASL. CoCASL allows algebraic and coalgebraic specification to be mixed. We have shown that the well-known coalgebraic modal logic and even the modal μ -calculus can easily be expressed in CoCASL, and give sufficient criteria for the existence of cofree models (also in the case of nested cofree and free specifications). Finally, we have shown how the existing coding of CASL into higher-order logic can be extended to CoCASL.

CoCASL is more expressive than other algebra-coalgebra combinations in the literature: [7] uses a simpler logic, while hidden algebra such as in BOBJ [24] and reachable-observable algebra such as in COL [4, 5] do not support cofree types (at least not at the level of basic specifications), which in particular means that corecursive definitions are not conservative. For example, a cogenerated specification of streams in COL with say, a *cons* operation, has also a model consisting of pairs (finite list, bit), where the finite list specifies the first part of the behaviour, and the bit specifies the (constant) behaviour afterwards. The corecursive definition of a flip stream (consisting of alternating zeros and ones) is then non-conservative.

By contrast, cofree types in CoCASL support a style of specification separating the basic process type (with its data sorts, observers and other operations) from further, derived operations defined on top of this in a conservative way. Note that this is not a purely theoretical question: programming languages such as Charity [8] and Haskell [9] support infinite datastructures that correspond to the infinite trees in the behaviour algebras, and one should be able to specify that as many infinite trees as needed for all programs over some datastructure expressible in these languages are present in the models of a specification. The Haskell semantics for lazy datastructures (at least for the non-left- \rightarrow -recursive case) indeed comprises *all* infinite trees, i.e. is captured exactly by a behaviour algebra.

Unlike COL [4, 5], CoCASL does not simultaneously support a glass-box and a black-box view on a specification. However, we plan to develop a notion of behavioural refinement between CoCASL specifications. Then, the black-box/glass-box view of [4] could be expressed in CoCASL as a refinement of a black-box specification into a glass-box one, thus also providing a clear separation of concerns.

The *Coalgebraic Class Specification Language* CCSL [25], developed in close cooperation with the LOOP project [29], is based on the observation of [22] that coalgebras can give a semantics to classes of object-oriented languages. CCSL provides a notation for parameterized class specifications based on final coalgebras. Its semantic is based on a higher-order equational logic and it provides theorem proving support by compilers that translate CCSL into the higher-order logic of PVS and Isabelle. In its current version, CCSL does not support data type specifications with partial constructors, axioms or equations, i.e. it only supports free types in the sense of CASL. Recently CCSL has been extended by binary methods [28], which are supported in CoCASL via cogenerated constraints.

At the level of proof principles, recent research about circular coinduction [10] and terminal sequence induction [21] is expected to provide tactics for the encoding of CoCASL into Isabelle/HOL.

Acknowledgements

The authors wish to thank Christoph Lüth for useful discussions, Erwin R. Catesbeiana for providing the larger perspective, and the participants of an informal CoCASL and observability meeting, Hubert Baumeister, Michel Bidoit, Rolf Hennicker, Bernd Krieg-Brückner, Don Sannella, Andrzej Tarlecki, and Martin Wirsing, as well as Alexander Kurz, for intensive feedback to a draft version of this work.

References

1. J. Adámek, H. Herrlich, and G. E. Strecker, *Abstract and concrete categories*, Wiley Interscience, 1990.
2. M. Arbib and E. Manes, *Parametrized data types do not need highly constrained parameters*, Inform. Control **52** (1982), 139–158.
3. M. Barr, *Terminal coalgebras in well-founded set theory*, Theoret. Comput. Sci. **114** (1993), 299–315.
4. M. Bidoit and R. Hennicker, *On the integration of observability and reachability concepts*, Fossacs, LNCS, vol. 2303, Springer, 2002, pp. 21–36.
5. Michel Bidoit, Rolf Hennicker, and Alexander Kurz, *Observational logic, constructor-based logic, and their duality*, Theoret. Comput. Sci. **298** (2003), 471–510.
6. P. Burmeister, *Partial algebras — survey of a unifying approach towards a two-valued model theory for partial algebras*, Algebra Universalis **15** (1982), 306–358.

7. C. Cirstea, *On specification logics for algebra-coalgebra structures: Reconciling reachability and observability*, LNCS **2303** (2002), 82–97.
8. R. Cockett and T. Fukushima, *About Charity*, Yellow Series Report 92/480/18, Univ. of Calgary, Dept. of Comp. Sci., 1992.
9. S. P. Jones et al., *Haskell 98: A non-strict, purely functional language*, (1999), <http://www.haskell.org/onlinereport>.
10. J. Goguen, K. Lin, and G. Rosu, *Conditional circular coinductive rewriting*, Automated Software Engineering, IEEE Press, 2000, pp. 123–131.
11. J. A. Goguen, *Hidden algebraic engineering*, Algebraic Engineering (Chrystopher Nehaniv and Masami Ito, eds.), World Scientific, 1999, pp. 17–36.
12. H. P. Gumm and T. Schröder, *Coalgebras of bounded type*, Math. Struct. Comput. Sci. **12** (2002), 565–578.
13. D. Kozen, *Results on the propositional mu -calculus*, Theoret. Comput. Sci. **27** (1983), 333–354.
14. A. Kurz, *Specifying coalgebras with modal logic*, Theoret. Comput. Sci. **260** (2001), 119–138.
15. ———, *Logics admitting final semantics*, Foundations of Software Science and Computation Structures, LNCS, vol. 2303, Springer, 2002, pp. 238–249.
16. T. Mossakowski, *CASL: From semantics to tools*, TACAS, LNCS, vol. 1785, Springer, 2000, pp. 93–108.
17. ———, *Relating CASL with other specification languages: the institution level*, Theoret. Comput. Sci. **286** (2002), 367–475.
18. T. Mossakowski, M. Roggenbach, and L. Schröder, *CoCASL at work — modelling process algebra*, CMCS 03, ENTCS, vol. 82(1), 2003.
19. T. Mossakowski, L. Schröder, M. Roggenbach, and H. Reichel, *Algebraic-coalgebraic specification in CoCASL*, Tech. report, University of Bremen, available at <http://www.informatik.uni-bremen.de/~lschrode/papers/cocas1.ev.ps>.
20. P. D. Mosses (ed.), *CASL — the common algebraic specification language. Reference manual*, Springer, to appear.
21. D. Pattinson, *Expressive logics for coalgebras via terminal sequence induction*, Tech. report, LMU München, 2002.
22. H. Reichel, *An approach to object semantics based on terminal co-algebras*, Math. Struct. Comput. Sci. **5** (1995), 129–152.
23. ———, *A uniform model theory for the specification of data and process types*, WADT, LNCS, vol. 1827, Springer, 2000, pp. 348–365.
24. G. Roşu, *Hidden logic*, Ph.D. thesis, Univ. of California at San Diego, 2000.
25. J. Rothe, H. Tews, and B. Jacobs, *The Coalgebraic Class Specification Language CCSL*, J. Universal Comput. Sci. **7** (2001), 175–193.
26. J. Rutten, *Universal coalgebra: A theory of systems*, Theoret. Comput. Sci. **249** (2000), 3–80.
27. A. Tarlecki, *On the existence of free models in abstract algebraic institutions*, Theoret. Comput. Sci. **37** (1985), 269–304.
28. H. Tews, *Coalgebraic methods for object-oriented languages*, Ph.D. thesis, Dresden Univ. of Technology, 2002.
29. J. van den Berg and B. Jacobs, *The LOOP compiler for Java and JML*, LNCS **2031** (2001), 299–312.