

# CoCASL

## Proof support for co-algebraic specification

---

Horst Reichel  
Dresden

Till Mossakowski, Markus Roggenbach, Lutz Schröder  
Bremen



---

# Motivation

- Add **process logic** to CASL

---

# Motivation

- Add **process logic** to CASL
- as an **easy** extension

---

# Motivation

- Add **process logic** to CASL
- as an **easy** extension
- with **good tool support!**

# Motivation

- Add **process logic** to CASL
- as an **easy** extension
- with **good tool support!**
- towards a standardized **combined algebraic-coalgebraic** language (based on well-known coalgebraic techniques).

# Coalgebra dualizes algebra

Algebra	Coalgebra
inductive datatypes	coinductive datatypes
constructors	selectors
generation	observability
<b>generated type</b>	<b>cogenerated type</b>
initial	final
<b>free type</b>	<b>cofree type</b>
<b>free { . . . }</b>	<b>cofree { . . . }</b>

## cogenerated types

**spec** BITSTREAM1 =

**free type**  $Bit ::= 0 \mid 1$

**cogenerated type**

$BitStream1 ::= (hd : Bit; tl : BitStream)$

**end**

Semantics:

equality is the largest congruence w.r.t. the introduced sorts and selectors

(full abstractness/coinduction principle)

BITSTREAM-models are subsets of  $Bit^{\mathbb{N}}$  (up to iso)

## cofree types

**spec** BITSTREAM2 =

**free type**  $Bit ::= 0 \mid 1$

**cofree type**

$BitStream ::= (hd : Bit; tl : BitStream)$

**end**

Semantics:

absolutely final coalgebra (consisting of all infinite trees)

BITSTREAM2-models are isomorphic to  $Bit^{\mathbb{N}}$

Main benefit: **corecursive definitions are always conservative!**

**cofree** { . . . }

**spec** BITSTREAM3 =

**free type** *Bit* ::= 0 | 1

**then cofree** {

**type** *BitStream* ::= (*hd* : *Bit*; *tl* : *BitStream*)

$\forall s : \textit{BitStream}$

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

**end**

Semantics: final coalgebra w.r.t. the axioms

BITSTREAM3-model (unique up to iso):

those bitstreams where two 0's are always followed by a 1.

## Modal logic (A. Kurz) as syntactic sugar

```

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

```

Semantics: the same as on the previous slide

BITSTREAM3-model (unique up to iso):  
 those bitstreams where two 0's are always followed by a 1.

## Existence of cofree models

From general results about coalgebra (Gumm, Aczel, Adámek, Kurz) we know:

The cofree coalgebra exists for specifications with

- selector-based datatype definitions
- also with initially specified types (e.g. lists, sets) as results of selectors (observers)
- with modal propositional logic formulas (and rules), and even  $\mu$ -calculus

## Example: free { . . . } within cofree { . . . }

```

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

```

# Fairness properties using cofree and free

```

spec BITSTREAM4 =
  free type Bit ::= 0 | 1
  then cofree {
    type BitStream ::= (hd : Bit; tl : BitStream)
    then free {
      pred fair : BitStream
      •  $fair \Leftrightarrow hd = 1 \vee \langle tl \rangle fair$ 
      • fair }
    }
  }

```

} corresponds to  $\mu$ -calculus formula

BITSTREAM4-model (unique up to iso):  
 those bitstreams where always eventually a 1 will come.

## Example: Moore automata

```
spec MOORE =  
  sorts In, Out  
  then cofree {  
    sort State  
    ops next :  $In \times State \rightarrow State$  }  
    observe :  $State \rightarrow Out$   
end
```

MOORE-model (unique up to iso): the final automaton?!

Problem: cofreeness holds only for homomorphisms being the identity on  $In$ .

## Fibre cofreeness . . .

- fibre-cofreeness = cofreeness w.r.t. restricted model categories
- the reduct of all homomorphisms w.r.t. a special signature morphism is the identity
- in  $\text{CASL}$ , this signature morphism is the inclusion of the input sorts
- input sorts = those sorts  $s_i$  with  $f : s_1, \dots, s_i, \dots, s_n \rightarrow s$ ,  $s$  a new sort,  $s_i$  not occurring as result sort of a new function.
- helps to deal with (otherwise higher-order) input params

# . . . is the solution

With fibre-cofreeness as semantics of **cofree** { . . . }, we get the expected semantics, namely the final coalgebra of

$$State \rightarrow State^{In} \times Out$$

## Tool support

- Extend HOL-CASL (encoding of CASL in Isabelle/HOL)
- **cogenerated types**: coinduction is a second-order principle
- **cofree types**: coinductive definitions and proofs are already provided by Isabelle/HOL
- **cofree**  $\{SP\}$  for  $SP$  flattenable with only modal axioms: just as with **cofree types**; here, we additionally have to restrict ourselves to those behaviours that satisfy the axioms

## Example: free { . . . } within cofree { . . . }

```

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

```

## Tool support for this

- 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
- obtain the cofree type over the free type by a quotient, following a result of Gumm and Schröder.

## cofree { . . . } within free { . . . }

A proof principle for free specifications containing cofree specifications seems to be much harder to obtain.

Suggestion: avoid the outer free specification and use a generation axiom plus some characterization of equality by suitably chosen observers instead!

**But notice:** algebraic-coalgebraic nestings are possible with  
*SP* then free {*SP*<sub>1</sub>} then cofree {*SP*<sub>2</sub>} then free . . .

## The modal $\mu$ -calculus

- $\mu$  is expressible with freely defined predicates
- $\nu$  is expressible with cofreely defined predicates
- nesting of  $\mu$  and  $\nu$  corresponds to nesting of **free** and **cofree**
- Proof support: nested least and greatest fixed points in Isabelle

## Conclusion

- CoCASL is a relatively easy extension of CASL
- algebraic and coalgebraic specification can be mixed
- CASL tool set has been extended to CoCASL
- Encoding to Isabelle/HOL can be extended
  - theory is clear, but implementation needs to be done!

---

## Related and future work

- algebra-coalgebra structures (Cîrstea) — simpler logic

---

## Related and future work

- algebra-coalgebra structures (Cîrstea) — simpler logic
- CCSL (Jacobs/Thews) — more OO?, fewer datatypes

## Related and future work

- algebra-coalgebra structures (Cîrstea) — simpler logic
- CCSL (Jacobs/Thews) — more OO?, fewer datatypes
- Expander (Padawitz) — ?

## Related and future work

- algebra-coalgebra structures (Cîrstea) — simpler logic
- CCSL (Jacobs/Thews) — more OO?, fewer datatypes
- Expander (Padawitz) — ?
- BOBJ (Goguen/Rosu), COL (Bidoit/Hennicker): do not support cofree types —  
but we should to add some notion of behavioural refinement

## Related and future work

- algebra-coalgebra structures (Cîrstea) — simpler logic
- CCSL (Jacobs/Thews) — more OO?, fewer datatypes
- Expander (Padawitz) — ?
- BOBJ (Goguen/Rosu), COL (Bidoit/Hennicker): do not support cofree types —  
but we should to add some notion of behavioural refinement
- Charity (Cockett), lazy Haskell (programming lanugages)

## Related and future work

- algebra-coalgebra structures (Cîrstea) — simpler logic
- CCSL (Jacobs/Thews) — more OO?, fewer datatypes
- Expander (Padawitz) — ?
- BOBJ (Goguen/Rosu), COL (Bidoit/Hennicker): do not support cofree types —  
but we should to add some notion of behavioural refinement
- Charity (Cockett), lazy Haskell (programming lanuguages)
- add circular coinduction (Goguen/Rosu) and terminal sequence induction (Pattinson) as tactics to Isabelle