CoCasl

Proof support for co-algebraic specification

Horst Reichel
Dresden

Till Mossakowski, Markus Roggenbach, Lutz Schröder
Bremen

WADT 2002
Motivation

- Add process logic to CASL
Motivation

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

• Add process logic to \texttt{CASL}
• as an \textit{easy} extension
• with \textit{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

<table>
<thead>
<tr>
<th>Algebra</th>
<th>Coalgebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>inductive datatypes</td>
<td>coinductive datatypes</td>
</tr>
<tr>
<td>constructors</td>
<td>selectors</td>
</tr>
<tr>
<td>generation</td>
<td>observability</td>
</tr>
<tr>
<td><strong>generated type</strong></td>
<td><strong>cogenerated type</strong></td>
</tr>
<tr>
<td>initial</td>
<td>final</td>
</tr>
<tr>
<td><strong>free type</strong></td>
<td><strong>cofree type</strong></td>
</tr>
<tr>
<td><strong>free { . . . }</strong></td>
<td><strong>cofree { . . . }</strong></td>
</tr>
</tbody>
</table>
cogenerated types

spec \textbf{BitStream1} =

free type \textit{Bit} ::= 0 | 1

cogenerated type

\textit{BitStream1} ::= (hd : Bit; tl : BitStream)

end

Semantics:
equality is the largest congruence w.r.t. the introduced sorts and selectors
(full abstractness/coinduction principle)

\textbf{BitStream}-models are subsets of \textit{Bit}^N (up to iso)
cofree types

spec \texttt{BitStream2} =
  \text{free type} \texttt{Bit} ::= 0 \mid 1
  \text{cofree type}
    \texttt{BitStream} ::= (\texttt{hd} : \texttt{Bit}; \texttt{tl} : \texttt{BitStream})

end

Semantics:
absolutely final coalgebra (consisting of all infinite trees)

\texttt{BitStream2}-models are isomorphic to \( \text{Bit}^\text{IN} \)

Main benefit: corecursive definitions are always conservative!
cofree \{ \ldots \} \\

\textbf{spec} \textbf{BitStream3} = \\
\textbf{free} \textbf{type} Bit ::= 0 | 1 \\
then cofree \{ \\
\textbf{type} BitStream ::= (hd : Bit; tl : BitStream) \\
\forall s : BitStream \\
\quad \bullet \ hd(s) = 0 \land hd(tl(s)) = 0 \Rightarrow hd(tl(tl(s)))) = 1 \} \\
end \\
Semantics: final coalgebra w.r.t. the axioms \\

\textbf{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 ∧ <tl> hd = 0
      ⇒ <tl><tl> 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 \{ . . . \}

\textbf{spec} \textsc{NonDeterministicAutomata} =

\begin{align*}
\text{sort} & \quad \text{In} \\
\text{then} & \quad \text{cofree} \{ \\
\text{sort} & \quad \text{State} \\
\text{then} & \quad \text{free} \{ \\
\text{type} & \quad \text{Set} ::= \{\} | \{\_\_\_\_\_\_\}(\text{State}) | \_\_ \cup \_\_((\text{Set}; \text{Set})) \\
\text{op} & \quad \_\_ \cup \_\_ : \text{Set} \times \text{Set} \to \text{Set}, \\
& \quad \text{assoc, comm, idem, unit} \{\} \\
\text{then} & \quad \text{op} \quad \text{next} : \text{In} \times \text{State} \to \text{Set} \\
\text{end}
\end{align*}
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 ⇔ hd = 1 ∨ <tl> fair
    • fair }
}
```

BitStream4-model (unique up to iso): those bitstreams where always eventually a 1 will come.
Example: Moore automata

\texttt{spec } \texttt{Moore =}

\texttt{sorts } \texttt{In, Out}

\texttt{then } \texttt{cofree } \{ 

\texttt{sort State}

\texttt{ops next : In } \times \texttt{ State } \to \texttt{ State } \}

\texttt{observe : State } \to \texttt{ Out}

\texttt{end}

\texttt{Moore}-model (unique up to iso): the final automaton?!

Problem: cofreeness holds only for homomorphisms being the identity on \texttt{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 Casl, this signature morphism is the inclusion of the input sorts
- input sorts = those sorts $s_i$ with $f : s_1, \ldots, s_i, \ldots, s_n \rightarrow s$, $s$ 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 \texttt{cofree \{ . . . \}}, we get the expected semantics, namely the final coalgebra of

\[ \text{State} \rightarrow \text{State}^{\text{In}} \times \text{Out} \]
Tool support

• Extend \texttt{HOL-CASL} (encoding of \texttt{CASL} in Isabelle/HOL)

• \textbf{cogenerated types}: coinduction is a second-order principle

• \textbf{cofree types}: coinductive definitions and proofs are already provided by Isabelle/HOL

• \textbf{cofree} \{\textit{SP}\} for \textit{SP} flattenable with only modal axioms: just as with \textbf{cofree types}; here, we additionally have to restrict ourselves to those behaviours that satisfy the axioms
Example: free \{ \ldots \} within cofree \{ \ldots \}

\textbf{spec} \textsc{NonDeterministicAutomata} =

\begin{verbatim}
  sort \textit{In}
  then cofree {
    sort \textit{State}
    then free {
      type \textit{Set} ::= \{\} | \{\ldots\}(\textit{State}) | \ldots \cup \ldots(\textit{Set}; \textit{Set})
      op \ldots \cup \ldots : \textit{Set} \times \textit{Set} \rightarrow \textit{Set},
                          \text{assoc, comm, idem, unit} \{\} \}
      then op \textit{next} : \textit{In} \times \textit{State} \rightarrow \textit{Set} \}
  end
\end{verbatim}
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.
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

\[ SP \text{ then free } \{ SP_1 \} \text{ then cofree } \{ SP_2 \} \text{ then free } \ldots \]
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

• \textit{CoCASL} is a relatively easy extension of \textit{CASL}

• algebraic and co-algebraic specification can be mixed

• \textit{CASL} tool set has been extended to \textit{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 lanuguages)
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)
• add circular coinduction (Goguen/Rosu) and terminal sequence induction (Pattinson) as tactics to Isabelle