Coalgebras and Temporal Logic

Bart Jacobs
Dep. Comp. Sci, Univ. Nijmegen
bart@cs.kun.nl
http://www.cs.kun.nl/~bart

Contents

In order to fix notation: the coproduct or disjoint union of two sets X, Y is:

$$X + Y \;=\; \{\langle x, 0\rangle \mid x \in X\} \;\cup\; \{\langle y, 1\rangle \mid y \in Y\}$$

with coprojections:

$$X \xrightarrow{\quad \kappa \quad} X + Y \xleftarrow{\quad \kappa' \quad} Y$$

and case or match construction: for functions $f\colon X \longrightarrow A$, $g\colon Y \longrightarrow Z$, there is a (unique) function $h\colon X + Y \longrightarrow A$ written as

$$\begin{array}{rl} z \longmapsto & \mathsf{CASES}\ z\ \mathsf{OF} \\ & \quad \kappa x \mapsto f(x) \\ & \quad \kappa' y \mapsto g(y) \\ & \mathsf{ENDCASES} \end{array}$$

with $h \circ \kappa = f$, $h \circ \kappa' = g$.

Correspondences

We shall work up-to the following bijective correspondences.

$$\frac{A \longrightarrow X \times Y}{A \longrightarrow X \qquad A \longrightarrow Y} \qquad\qquad \frac{X + Y \longrightarrow A}{X \longrightarrow A \qquad Y \longrightarrow A}$$

$$\frac{X \longrightarrow Y^A}{X \times A \longrightarrow Y} \qquad\qquad \frac{P \hookrightarrow X}{X \longrightarrow \{0, 1\}}$$

(Which are crucial in describing the category of sets as a topos.)

Consider the following fragment from a coalgebraic specification of (possibly infinite) binary trees.

```
BEGIN BinaryTree[ A : Type ] :  CLASSSPEC
    METHOD children :  Self -> 1 + (Self × Self);
           label :  Self -> A;
           elem :  Self × A -> bool;
           ...
    ASSERTION
        naive_elem_def :  FORALL(a:A) : elem(x, a) <=> (label(x) = a) OR
                              CASES children(x) OF
                                κy |-> false
                                κ'(x1, x2) |-> elem(x1, a) OR elem(x2, a)
                              ENDCASES
        ...
END BinaryTree
```

Binary trees, continued

But the set $A^{\{0,1\}^\star}$ of all infinite binary A-labelled trees forms a model of the specification with, for $x \in A^{\{0,1\}^\star}$,

$$\begin{array}{rcl} \mathtt{children}(x) & = & \kappa'\big(\lambda\alpha \in \{0,1\}^\star.\ x(0\cdot\alpha),\ \lambda\alpha \in \{0,1\}^\star.\ x(1\cdot\alpha)\big) \\ \mathtt{label}(x) & = & x(\langle\rangle) \\ \mathtt{elem}(x,a) & = & \mathtt{true}\ !!! \end{array}$$

What we would like to be able to define instead is:

$$\mathtt{elem}(x,a) \iff \big(\text{there is a successor tree } y \text{ of } x \text{ with } \mathtt{label}(y) = a\big)$$

That is:

$$\mathtt{elem}(x,a) \iff \Diamond\big(\lambda y.\ \mathtt{label}(y) = a\big)(x)$$

Recall the methods from the BinaryTree specification:

```
label : Self -> A;
children : Self -> 1 + (Self × Self);
elem : Self × A -> bool;
```

They determine an interface type

$$T(X) \;=\; A \times \big(1 + (X \times X)\big) \times \mathsf{bool}^A$$

In general, we consider interface types:

$$T \;:=\; X \mid A \mid T + T \mid T \times T \mid T^A \mid \mathcal{P}T$$

A coalgebra is a function of the form

$$X \xrightarrow{\quad c \quad} T(X)$$

where $T(X)$ is an interface type. X is called the state space.

Examples

- A coalgebra

$$X \xrightarrow{\qquad c \qquad} A \times \big(1 + (X \times X)\big) \times \mathsf{bool}^A$$

  for the BinaryTree interface type combines the three methods label, children, elem in a single function.

- A coalgebra

$$X \xrightarrow{\qquad c \qquad} \mathcal{P}(A \times X)$$

  describes a transition system with labels from A: put $x \xrightarrow{a} y$ for $(a, y) \in c(x)$.

- A coalgebra

$$X \xrightarrow{\qquad c \qquad} X^A \times \{0, 1\}$$

  describes a deterministic automaton with alphabet A: write $x \xrightarrow{a} y$ for $\pi(c(x))(a) = y$, and $x \downarrow$ for $\pi'(c(x)) = 1$.

Invariants

A predicate $P \subseteq X$ is an invariant for a coalgebra $c : X \longrightarrow T(X)$ if:

$$\forall x \in X.\, P(x) \Longrightarrow P^T(c(x))$$

where the predicate lifting $P^T \subseteq T(X)$ is defined by induction on the structure of T:

$$\begin{array}{rcl}
P^X & = & P \\
P^A & = & A \\
P^{T_1 \times T_2} & = & \{(x_1, x_2) \mid P^{T_1}(x_1) \;\wedge\; P^{T_2}(x_2)\} \\
P^{T_1 + T_2} & = & \{\kappa\, x_1 \mid P^{T_1}(x_1)\} \;\cup\; \{\kappa'\, x_2 \mid P^{T_2}(x_2)\} \\
P^{T^A} & = & \{f \mid \forall a \in A.\, P^{T_1}(f(a))\} \\
P^{\mathcal{P}T} & = & \{U \mid \forall z \in U.\, P^T(z)\}.
\end{array}$$

Thus, P is an invariant for a coalgebra c means that c maintains P.

Definition

With respect to a class (coalgebra) c, define for a prediate P on the state space of c,

$$(\Box P)(x) \iff \exists Q.\ \big(Q \text{ is an invariant for } c \ \wedge\ Q \subseteq P \ \wedge\ Q(x)\big)$$

And:

$$\Diamond P \ = \ \neg\Box\neg P.$$

Then:

- $\Box P$ is the greatest invariant contained in P. It is henceforth P.
- $\Diamond P$ holds for those states that are reachable from P, expressing eventually P.

The definitions of invariant, $\Box$ and $\Diamond$ are tailor-made for each class specification, because they are fully determined by the interface type/functor of the specification.

BinaryTree example

With the machinery of interface functors, predicate lifting, invariants and temporal operators, we return to the BinaryTree specification.

Now we can define elem: $\mathsf{Self} \times A \longrightarrow \mathsf{bool}$ as:

$$\mathtt{elem}(x, a) \Longleftrightarrow \Diamond\big(\lambda y.\ \mathtt{label}(y) = a\big)(x)$$

This definition is at the right level of abstraction. It would be precisely the same for Ternary trees, with operation

$$\mathsf{children}\colon \mathsf{Self} \longrightarrow 1 + \big(\mathsf{Self} \times \mathsf{Self} \times \mathsf{Self}\big)$$

because $\Diamond$ would change accordingly, again with meaning: for some future state ...

BinaryTree example

Adding $\Box$ and $\Diamond$ greatly increase the expressive power of the logic. But more could be asked.

The least μ and greatest ν fixed point operators on monotone functions on predicates further increase the expressiveness. With these operators from the μ-calculus we can express what it means for a binary tree to have finite or infinite behaviour:

$$
\begin{array}{rcl}
\mathsf{finite} & = & \mu P.\, \big(\lambda x.\ \ \mathsf{CASES}\ \mathsf{children}(x)\ \mathsf{OF} \\
& & \qquad\qquad \kappa y \mapsto \mathsf{true} \\
& & \qquad\qquad \kappa'(x_1, x_2) \mapsto P(x_1)\ \wedge\ P(x_2) \\
& & \qquad\quad\ \mathsf{ENDCASES}\,\big)
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{infinite} & = & \nu P.\, \big(\lambda x.\ \ \mathsf{CASES}\ \mathsf{children}(x)\ \mathsf{OF} \\
& & \qquad\qquad \kappa y \mapsto \mathsf{false} \\
& & \qquad\qquad \kappa'(x_1, x_2) \mapsto P(x_1)\ \vee\ P(x_2) \\
& & \qquad\quad\ \mathsf{ENDCASES}\,\big)
\end{array}
$$

```
BEGIN Queue[ A : Type ] :  CLASSSPEC
    METHOD size :  Self -> int;
           push :  Self × A -> Self;
           pop :  Self -> 1 + (A × Self);
    ASSERTION
       size_push :  FORALL(a:A) : size(push(x, a)) = size(x) + 1
       size_pop :  CASES pop(x) OF
                     κ∗ :  size(x) = 0,
                     κ′(a, y) :  size(y) = size(x) - 1
                   ENDCASES
       pop_push :  FORALL(a:A) :
           CASES pop(push(x, a)) OF
              κ∗ :  FALSE
              κ′(b, y) :  CASES pop(x) OF
                             κ∗ :  b = a AND y ↔ x
                             κ′(c, z) :  b = c AND y ↔ push(z, a)
           ENDCASES ENDCASES
    CONSTRUCTOR empty :  Self
    CREATION pop_empty :  pop(empty) = κ∗
END Queue
```

Queue consequences

We wish to prove the safety and progress properties:

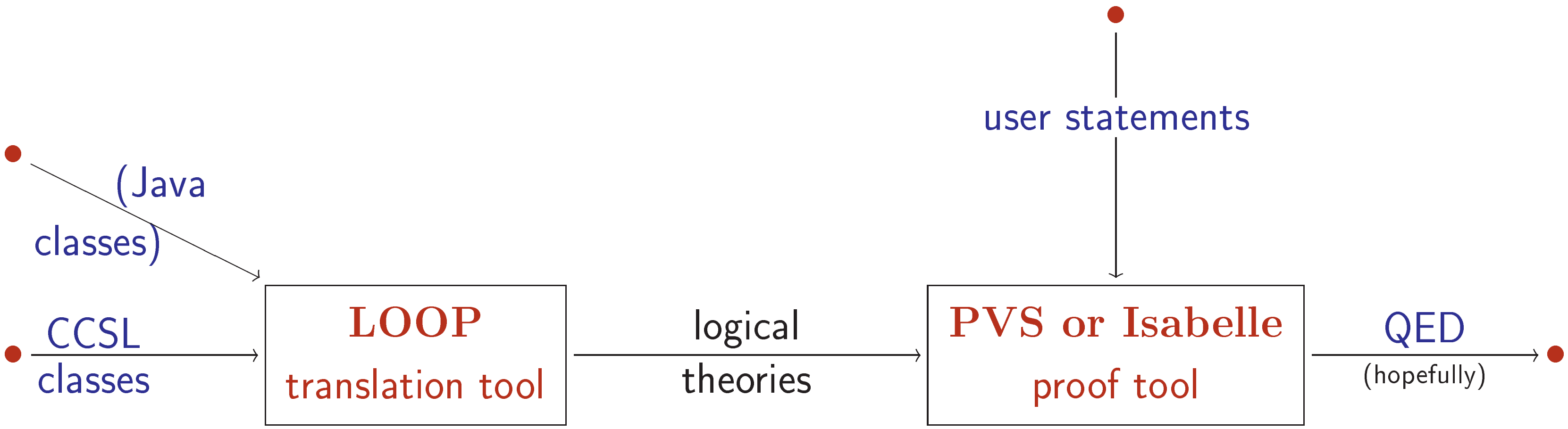
$$\Box\Big(\lambda y : \mathsf{Self} : \mathsf{size}(y) \geq 0 \wedge (\mathsf{size}(y) = 0 \Rightarrow \mathsf{pop}(y) = \kappa *)\Big)(\mathsf{empty})$$

And:

$$\Box\Big(\lambda y : \mathsf{Self}.\ \mathsf{size}(y) \geq 0 \wedge \Diamond\big(\lambda z : \mathsf{Self}.\ \mathsf{size}(z) = 0\big)(y)\Big)(\mathsf{empty})$$

The proofs use that the following is a Queue invariant:

$$\begin{array}{l} \mathsf{size}(x) \geq 0 \\ \wedge \\ \forall n < \mathsf{size}(x).\ \mathsf{iterate}(\mathsf{pop}, n)(x) \neq \kappa * \\ \wedge \\ \mathsf{iterate}(\mathsf{pop}, \mathsf{size}(x))(x) = \kappa * . \end{array}$$



- CCSL, for Coalgebraic Class Specification Language is a formal language for writing specifications of classes, involving parameterisation (by types and values), inheritance, aggregation.
- The LOOP tool provides a coalgebraic semantics in the logic of the back-end proof tool. It automatically generates tailor-made definitions of invariant, bisimulation, $\Box$ and $\Diamond$ for the class that is translated.
- The LOOP tool is implemented in Ocaml and has an Emacs interface.

Nexttime and lasttime

Earlier we saw for an interface functor T the predicate lifting operation:

$$\big(P \subseteq X\big) \longmapsto \big(P^T \subseteq T(X)\big)$$

This operation has a left adjoint

$$\big(Q \subseteq T(X)\big) \longmapsto \big(Q_T \subseteq X\big)$$

which can also be defined by induction on T. We thus have a Galois connection:

$$Q_T \;\subseteq\; P \quad \Longleftrightarrow \quad Q \subseteq P^T$$

For a coalgebra $c : X \longrightarrow T(X)$ and a predicate $P \subseteq X$, define one-step NextTime and LastTime operators:

$$\underrightarrow{c}\,P \;=\; \{x \in X \mid P^T(c(x))\}$$

$$\underleftarrow{c}\,P \;=\; \big(\{c(x) \mid x \in P\}\big)_T$$

The Galois connection $(-)_T \dashv (-)^T$ yields another Galois connection between NextTime and LastTime:

$$\underleftarrow{c}\,P \subseteq Q \quad\Longleftrightarrow\quad P \subseteq \underrightarrow{c}\,Q$$

on the poset $\mathcal{P}X$ of predicates on the state space X of a coalgebra $c\colon X \to T(X)$.

Definition [von Karger 98]
A Galois algebra is a complete Boolean algebra with a Galois connection $\underleftarrow{\bullet} \dashv \underrightarrow{\bullet}$.

Proposition
Every coalgebra of a polynomial functor yields a Galois algebra: $c\colon X \to T(X)$ yields $(\mathcal{P}X, \underrightarrow{c})$.

In an arbitrary Galois algebra one interprets, for example,

$$
\begin{array}{rcl}
\underset{\rightarrow}{\bullet} P & = & \text{weak nexttime } P \\
\underset{\leftarrow}{\bullet} P & = & \text{strong lasttime } P \\
\neg \underset{\rightarrow}{\bullet} \neg P & = & \text{strong nexttime } P \\
\neg \underset{\leftarrow}{\bullet} \neg P & = & \text{weak lasttime } P \\
\underset{\Rightarrow}{c} P = \Box P & = & \nu S.\, (P \wedge \underset{\rightarrow}{c} S) \;=\; \text{weak henceforth } P \\
\underset{\Leftarrow}{c} P & = & \mu S.\, (P \vee \underset{\leftarrow}{c} S) \;=\; \text{strong earlier } P
\end{array}
$$

etc.

Von Karger proves that all axioms and rules of Computation Tree Logic (CTL) are valid in Galois algebras, and that all axioms and rules of Linear Temporal Logic (LTL) are valid in Galois algebras satisfying certain linearity conditions.

Examples

- For a Queue coalgebra $c = \langle \mathsf{size}, \mathsf{push}, \mathsf{pop} \rangle : X \longrightarrow \mathsf{int} \times \mathsf{Self}^A \times \big(1 + (A \times \mathsf{Self})\big)$,

$$
\begin{aligned}
\big(\underset{\rightarrow}{c}\, P\big)(x) \iff \; & \forall a \in A.\ P(\mathsf{push}(x, a)) \wedge \\
& \forall a \in A, y \in X.\ \mathsf{pop}(x) = \kappa'(a, y) \Rightarrow P(y) \\
\big(\underset{\leftarrow}{c}\, P\big)(x) \iff \; & \exists a \in A, y \in X.\ \mathsf{push}(y, a) = x \;\wedge\; P(y) \\
& \vee\ \exists a \in A, y \in X.\ \mathsf{pop}(y) = \kappa'(a, x) \;\wedge\; P(y)
\end{aligned}
$$

- For a transition system $c : X \to \mathcal{P}(A \times X)$,

$$
\begin{aligned}
\big(\underset{\rightarrow}{c}\, P\big)(x) \iff \; & \forall a \in A, y \in X.\ x \xrightarrow{a} y \Rightarrow P(y). \\
\big(\underset{\leftarrow}{c}\, P\big)(x) \iff \; & \exists a \in A, y \in X.\ y \xrightarrow{a} x \;\wedge\; P(y).
\end{aligned}
$$

Thus we can apply the theory of Galois algebras for every class/coalgebra.

A homomorphism between coalgebras $\big(X \xrightarrow{c} T(X)\big)$ and $\big(Y \xrightarrow{d} T(Y)\big)$ is a map $f\colon X \longrightarrow Y$ with:

$$
\begin{tikzcd}
T(X) \arrow[r, "T(f)"] & T(Y) \\
X \arrow[u, "c"] \arrow[r, "f"'] & Y \arrow[u, "d"']
\end{tikzcd}
$$

The induced inverse image function $f^{-1}\colon \mathcal{P}(Y) \longrightarrow \mathcal{P}(X)$ commutes with NextTime, but not with LastTime.

By defining an appropriate notion of homomorphism of Galois algebras we get a functor

$$\mathrm{CoAlg}(T)^{\mathrm{op}} \longrightarrow \mathbf{GA}$$

given by

$$\Big(X \xrightarrow{c} T(X)\Big) \longmapsto \Big(\mathcal{P}(X), \underset{\rightarrow}{c}\Big) \qquad \text{and} \qquad f \longmapsto f^{-1}$$

Several examples of “logics for coalgebras” arise in this manner.

Conclusions

Coalgebras and temporal logic mix very well:

- Coalgebras form a rich source of models for temporal logics.
- The temporal operators $\Box$ and $\Diamond$ are very convenient for reasoning about state-based systems.

Future work

- Model checking for coalgebras
- Proof-theoretic properties of temporal logics, parametrised by polynomial functors.

- B. Jacobs, Invariants, Bisimulations and the Correctness of Coalgebraic Refinements. AMAST'97, LNCS 1349, 1997.
- B. Jacobs, The temporal logic of coalgebras via Galois algebras, Techn. Rep. CSI-R9906, Comput. Sci. Inst., Univ. of Nijmegen, 1999.
- B. Jacobs, Exercises in Coalgebraic Specifications. Manuscript.

See also `http://www.cs.kun.nl/~bart`

- B. von Karger, Temporal algebra. *Math. Struct. in Comp. Sci.* (8), 1998, p. 277–320.
- L.S. Moss. Coalgebraic logic. *Ann. Pure & Appl. Logic*, 1999, to appear.
- M. Rößiger. From modal logic to terminal coalgebras. *Theor. Comp. Sci.*, 1999, to appear.
- A. Kurz. Specifying coalgebras with modal logic. *Coalgebraic Methods in Computer Science*, ENTCS 11, 1998.
- A. Sernadas, C. Sernadas, and J.F. Costa. Object specification logic. *Journ. of Logic and Computation*, 5(5), 1995, p. 603–630.