Assertional and Behavioural Refinement in Coalgebraic Specification

Bart Jacobs
Department of Computer Science,
University of Nijmegen
Email: bart@cs.kun.nl
URL: http://www.cs.kun.nl/~bart

Joint work with:

Hendrik Tews

Contents

Why coalgebras?

In state-based computing state-transformer functions are important.

Often, such functions have different output modes. E.g. statements in Java:

$$\mathsf{Self} \xrightarrow{\mathsf{stat}} \mathsf{unit} + \mathsf{Self} + \mathsf{Abrupt}$$

Or transition functions of labeled transition systems

$$\mathsf{Self} \xrightarrow{\mathsf{trans}} \mathcal{P}(\mathsf{Label} \times \mathsf{Self})$$

In general, a coalgebra is a function of the form:

$$\mathsf{Self} \longrightarrow \boxed{\quad \cdots \quad \mathsf{Self} \quad \cdots \quad}$$

with a structured type as codomain. This type is the interface of the coalgebra.

Coalgebras are useful for describing various forms of computations: with non-determinism, partiality, exceptions, ... These different forms are reflected by the different codomain types.

Dually, an algebra can be described as a function of the form:

$$\boxed{\quad \cdots \quad \mathsf{Self} \quad \cdots \quad} \longrightarrow \mathsf{Self}$$

Coalgebras are used for modeling state-based systems.

Temporal and modal logic are often used for reasoning about such systems, via operators like:

- "for all future/past states" ...
- "for some future/past states" ...

Therefore, one expects a connection. It wil be a side topic of this talk.

But first, we concentrate on interface types.

Polynomial / Interface functors include:

- The identity functor;
- The constant functors $X \longmapsto A$, for an arbitrary set A;
- The product $X \longmapsto \mathcal{T}_1(X) \times \mathcal{T}_2(X)$ of two polynomial functors $\mathcal{T}_1, \mathcal{T}_2$;
- The coproduct $X \longmapsto \mathcal{T}_1(X) + \mathcal{T}_2(X)$ of two polynomial functors $\mathcal{T}_1, \mathcal{T}_2$;
- The exponent $X \longmapsto \mathcal{T}_1(X)^A$ for a polynomial functors $\mathcal{T}_1$ and an arbitrary set A.

Example

$$X \longmapsto 1 + (A + X)^B + (X \times X)^C.$$

Let $\mathcal{T}$ be a polynomial functor. An algebra consists of a set A together with a function

$$\mathcal{T}(A) \xrightarrow{\quad a \quad} A$$

A coalgebra consists of a set C with a function

$$C \xrightarrow{\quad c \quad} \mathcal{T}(C)$$

Example

- The unit u, multiplication m, and inverse operation i of a group G form an algebra

$$1 + (G \times G) + G \xrightarrow{[u, m, i]} G$$

for the functor $\mathcal{T}(X) = 1 + (X \times X) + X$.

- The head h and tail t operation on the set A^ω of infinite sequences of elements from a set A form a coalgebra

$$A^\omega \xrightarrow{\langle h, t \rangle} A \times A^\omega$$

for the functor $\mathcal{T}(X) = A \times X$.

A coalgebraic specification is a description of a coalgebra with an initial state, satisfying certain assertions. Formally:

Definition

A coalgebraic specification $(\mathcal{T}, E, I)$ consists of

- $\mathcal{T} = \mathcal{T}(\mathsf{Self})$ is a polynomial functor involving one special type symbol Self,
- $E = E(c)$ and $I = I(c)$ are (syntactic) predicates on the state space Self of a special $\mathcal{T}$-coalgebra symbol $c\colon \mathsf{Self} \to \mathcal{T}(\mathsf{Self})$.

Consider a simple negator system with a unit delay: the result of a negation request only becomes visible after a second request.
The intermediate value is handled via a "private" auxiliary operation.

BEGIN *DelayAuxNeg* : **CLASSSPEC**
  METHOD
    val: Self $\longrightarrow 2$ // $2 = \{0, 1\}$
    neg: Self $\times\, 2 \longrightarrow$ Self
  PRIVATE METHOD   // ignore 'PRIVATE' at first
    aux: Self $\longrightarrow 2$
  ASSERTION
    $\forall b \in 2.\ \mathsf{aux}(\mathsf{neg}(x, b)) = \neg b$
    $\forall b \in 2.\ \mathsf{val}(\mathsf{neg}(x, b)) = \mathsf{aux}(x)$
  CREATION
    $\mathsf{aux}(\mathsf{new}) = 0$
    $\mathsf{val}(\mathsf{new}) = 0$
END *DelayAuxNeg*

The *DelayAuxNeg* example consists of $(\mathcal{T}, E, I)$ with:

- The polynomial functor $\mathcal{T}$ is given by:

$$\mathcal{T}(\mathsf{Self}) \;=\; 2 \times \mathsf{Self}^2 \times 2$$

and a $\mathcal{T}$-coalgebra combines the three methods in a single function:

$$c = \langle \mathsf{val}, \mathsf{neg}, \mathsf{aux} \rangle : \mathsf{Self} \longrightarrow \mathcal{T}(\mathsf{Self})$$

- The predicate $E(c)$ on Self, for $c = \langle \mathsf{val}, \mathsf{neg}, \mathsf{aux} \rangle$ is given by the assertion:

$$\begin{aligned} E(c) \;=\; & (\forall b \in 2.\ \mathsf{aux}(\mathsf{neg}(x, b)) = \neg b) \;\wedge \\ & (\forall b \in 2.\ \mathsf{val}(\mathsf{neg}(x, b)) = \mathsf{aux}(x)) \end{aligned}$$

- Similarly, the predicate $I(c)$ on Self is:

$$I(c) \;=\; (\mathsf{aux}(\mathsf{new}) = 0) \;\wedge\; (\mathsf{val}(\mathsf{new}) = 0)$$

where $\mathsf{new}\colon \mathsf{Self}$ is a special variable for an initial state.

Given a coalgebraic specification $(\mathcal{T}, E, I)$ and a concrete coalgebra $d\colon X \to \mathcal{T}(X)$ with an element $x \in X$, we write:

- $d, x \models E$ if x is in the interpretation $[\![\, E \,]\!](d) \subseteq X$ of E, in which d is used as interpretation of the coalgebra symbol c in $E = E(c)$;
- $d \models E$ if $d, x \models E$ for all $x \in X$. This will also be written as $d\colon X \stackrel{E}{\to} \mathcal{T}(X)$.

A model of a coalgebraic specification $(\mathcal{T}, E, I)$ consists of a coalgebra $d\colon X \to \mathcal{T}(X)$ with an initial state $x_0 \in X$ such that $d \models E$ and $d, x_0 \models I$.

Thus a model is an element of the disjoint union:

$$\coprod_{d\colon X \stackrel{E}{\to} \mathcal{T}(X)} \{x \in X \mid d, x \models I\} \quad \text{or, more briefly:} \quad \coprod_{d\colon X \stackrel{E}{\to} \mathcal{T}(X)} I(d).$$

As a model of the *DelayAuxNeg* specification, one can take:

- State space $X = 2 \times 2$
- Coalgebra $\langle \mathsf{val}, \mathsf{neg}, \mathsf{aux} \rangle \colon X \to 2 \times X^2 \times 2$ given by:

$$\begin{array}{rcl} X & \xrightarrow{\mathsf{val}} & 2 \\ (b_1, b_2) & \longmapsto & b_1 \end{array} \qquad \begin{array}{rcl} X \times 2 & \xrightarrow{\mathsf{neg}} & X \\ ((b_1, b_2), b) & \longmapsto & (b_2, \neg b) \end{array} \qquad \begin{array}{rcl} X & \xrightarrow{\mathsf{aux}} & 2 \\ (b_1, b_2) & \longmapsto & b_2 \end{array}$$

- Initial state $(0, 0) \in X$.

It is not hard to see that the *DelayAuxNeg*-assertions are satisfied.

Actually, this coalgebra is the final one satisfying the assertions: if also $d \colon Y \to 2 \times Y^2 \times 2$ satisfies the assertions, there is a unique homomorphism

$$f \colon Y \longrightarrow 2 \times 2 \qquad \text{namely} \qquad f = \langle \pi_1 \circ d, \pi_3 \circ d \rangle = \langle \mathsf{val}_d, \mathsf{aux}_d \rangle.$$

Bisimilarity

Coalgebras are dynamical systems consiting of a black-box state space X together with a "transition map" $X \longrightarrow \mathcal{T}(X)$ for observation and modification.

Equality of states is often not appropriate in coalgebra, but indistinguishability, or equality as far as we can observe, is.

This is called bisimilarity, and written as $\underline{\leftrightarrow}$.

Bisimulation

Bisimilarity is defined as the greatest bisimulation.

A bisimulation for two coalgebra $c: X \longrightarrow \mathcal{T}(X), d: Y \longrightarrow \mathcal{T}(Y)$ is a relation $R \subseteq X \times Y$ on the state spaces which is closed under c and d.

There are several ways to formalise this. We choose the logical approach (using induction on $\mathcal{T}$), since it is most convenient in reasoning.

Definition

For coalgebras $c\colon X \longrightarrow \mathcal{T}(X)$, $d\colon Y \longrightarrow \mathcal{T}(Y)$ a relation $R \subseteq X \times Y$ is a bisimulation if

$$(x,y) \in R \implies (c(x), d(y)) \in \mathsf{Rel}(\mathcal{T})(R)$$

where the relation lifting $\mathsf{Rel}(\mathcal{T})(R) \subseteq \mathcal{T}(X) \times \mathcal{T}(Y)$ is defined by induction:

$\mathcal{T}$	$\mathsf{Rel}(\mathcal{T})(R)(z,w) \quad \text{for} \quad z\colon \mathcal{T}(X), w\colon \mathcal{T}(Y)$
X	$R(z,w)$
C	$z =_C w$
$\mathcal{T}_1 \times \mathcal{T}_2$	$\mathsf{Rel}(\mathcal{T}_1)(R)(\pi\, z, \pi\, w) \ \wedge\ \mathsf{Rel}(\mathcal{T}_2)(R)(\pi'\, z, \pi'\, w)$
$\mathcal{T}_1 + \mathcal{T}_2$	$\begin{cases} \text{if } (z,w) = (\kappa\, z_1, \kappa\, w_1) & \mathsf{Rel}(\mathcal{T}_1)(R)(z_1, w_1) \\ \text{if } (z,w) = (\kappa'\, z_2, \kappa'\, w_2) & \mathsf{Rel}(\mathcal{T}_2)(R)(z_2, w_2) \\ \text{otherwise} & \bot \end{cases}$
$\mathcal{T}_1^A$	$\forall a : A\, .\, \mathsf{Rel}(\mathcal{T}_1)(R)(z(a), w(a))$

Intuition

For $R \subseteq X \times Y$, the lifting $\mathsf{Rel}(\mathcal{T})(R) \subseteq \mathcal{T}(X) \times \mathcal{T}(Y)$ relates:

- all occurrences of the state space X and Y by R
- all occurrences of a constant C by equality.

And it does so in a "deep" manner:

$$
\begin{array}{rcl}
\mathcal{T}(X) & = & \boxed{\;\cdots \quad X \quad \cdots \quad C \quad \cdots \quad X \quad \cdots\;} \\[2ex]
\mathsf{Rel}(\mathcal{T})(R) & = & \qquad\quad \Big| R \qquad\quad \Big\| \qquad\quad \Big| R \\[2ex]
\mathcal{T}(Y) & = & \boxed{\;\cdots \quad Y \quad \cdots \quad C \quad \cdots \quad Y \quad \cdots\;}
\end{array}
$$

Definition

Consider two states $x \in X, y \in Y$ of coalgebras $c\colon X \longrightarrow \mathcal{T}(X), d\colon Y \longrightarrow \mathcal{T}(Y)$.

$$x \;\; {}_{c}\underline{\leftrightarrow}_{d} \;\; y \;\; \stackrel{\text{def}}{\Longleftrightarrow} \;\; \exists R.\, (R \text{ is a bisimulation for } c, d \;\wedge\; R(x, y))$$

Then:

- $\underline{\leftrightarrow}$ is the greatest bisimulation. This gives the coinduction proof principle:

  $\boxed{\text{In order to prove } x \,\underline{\leftrightarrow}\, y \text{ it suffices to find a bisimulation } R \text{ with } R(x, y)}$

- bisimilar states are observationally indistinguishable

The definition of $\underline{\leftrightarrow}$ is fully determined by the interface type/functor of a specification. Therefore it is tailor-made for each class specification.

Informally

In the theory of dynamical systems, an invariant is a predicate on the state space, which, once it holds, continues to hold no matter which operations are applied.

Briefly, all operations maintain an invariant.

Importance

- Invariants express safety properties of specifications.

  E.g. that a certain integer-valued attribute will always be non-negative.

- Invariants are used for the definition of the temporal operators $\Box$ and $\Diamond$.
- Invariants are needed for refinements.

Predicate lifting 1

For a coalgebra $c\colon X \longrightarrow \mathcal{T}(X)$ a predicate $P \subseteq X$ is an invariant if

$$x \in P \implies c(x) \in \mathsf{Pred}(\mathcal{T})(P)$$

where the predicate lifting $\mathsf{Pred}(\mathcal{T})(P) \subseteq \mathcal{T}(X)$ is defined by induction:

$\mathcal{T}$	$\mathsf{Pred}(\mathcal{T})(P)(z)$ for $z : \mathcal{T}(X)$
X	$P(z)$
C	$\top$
$\mathcal{T}_1 \times \mathcal{T}_2$	$\mathsf{Pred}(\mathcal{T}_1)(P)(\pi\ z) \ \wedge\ \mathsf{Pred}(\mathcal{T}_2)(P)(\pi'\ z)$
$\mathcal{T}_1 + \mathcal{T}_2$	$\begin{cases} \text{if } z = \kappa\, z_1 & \mathsf{Pred}(\mathcal{T}_1)(P)(z_1) \\ \text{if } z = \kappa'\, z_2 & \mathsf{Pred}(\mathcal{T}_2)(P)(z_2) \end{cases}$
$\mathcal{T}_1^A$	$\forall x : A\ .\ \mathsf{Pred}(\mathcal{T}_1)(P)(z(x))$

Intuition

For $P \subseteq X$, the lifting $\mathsf{Pred}(\mathcal{T})(P) \subseteq \mathcal{T}(X)$ says that P should hold at all occurrences of the state space X:

$$
\begin{array}{rcccccccc}
\mathcal{T}(X) = & \cdots & X & \cdots & C & \cdots & X & \cdots \\
 & & \Big| & & & & \Big| & \\
\mathsf{Pred}(\mathcal{T})(P) = & & \Big| & & & & \Big| & \\
 & & P & & & & P &
\end{array}
$$

Proposition

1. In the present set-theoretic setting, invariants P can be identified with sub-coalgebras:

$$
\begin{tikzcd}
\mathcal{T}(P) \arrow[r, hook] & \mathcal{T}(X) \\
P \arrow[u] \arrow[r, hook] & X \arrow[u, "c"']
\end{tikzcd}
$$

2. Predicate lifting $\mathrm{Pred}(\mathcal{T})\colon \mathcal{P}(X) \longrightarrow \mathcal{P}(\mathcal{T}(X))$ preserves arbitrary intersections.

   (Or, equivalently, it has a left Galois adjoint $\mathcal{P}(\mathcal{T}(X)) \longrightarrow \mathcal{P}(X)$.)

3. Invariants are closed under $\wedge$ and $\forall$:
   - If P and Q are invariants then so is $P \wedge Q$.
   - If $P(i)$ is an invariant for each $i \in I$, then so is $\forall i \in I.\, P(i)$.

Definition

With respect to a coalgebra $c \colon X \longrightarrow T(X)$, define for a prediate $P \subseteq X$,

$$(\Box P)(x) \stackrel{\text{def}}{\Longleftrightarrow} \exists Q.\, (Q \text{ is an invariant for } c \;\wedge\; Q \subseteq P \;\wedge\; Q(x))$$

And:

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

Then:

- $\Box P$ is the greatest invariant contained in P. It is an interior operator, expressing henceforth P.
- $\Diamond P$ holds for those states that are reachable from P. Thus it is a closure operator 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.

Rôle of refinements / implementations

- A refinement is a relative model of one "abstract" specification assuming a model of a "concrete" specification, using the same state space. Sometimes it is also called an incomplete program.
- Typically, the concrete specification has more implementation details and less underspecification than the abstract one.
- Refinements are important in the step-wise construction of an implementation:

$$\boxed{\text{Requirement Spec}} \dashrightarrow \boxed{\text{Spec}_1} \xrightarrow{\text{refinements}} \boxed{\text{Spec}_n} \dashrightarrow \boxed{\text{Program code}}$$

Assertional and behavioural refinement 1

In a refinement, a "model" of the concrete specification is turned into a "model" of the abstract one.

There are two basic views on "model"

- the assertional view: after translation, the abstract assertions should be satisfied.

  Slightly simplified: refinement is implication, or inclusion between classes of models.

- the behavioural one: the translated concrete model should behave as, or be indistinguishable from, an abstract one.

  Typically expressed via: (bi)similarity or trace-inclusion.

Assertional and behavioural refinement 2

Assertional refinement	Behavioural refinement
data / algebra refinement (Hoare, Goguen, Tarlecki, Sannella, Wirsing . . .)	action refinement (Gorrieri, Degano, Hennessey, van Glabbeek . . .)
program refinement (Back, . . .)	automata / transition system refinement (Lynch & Vaandrager, . . .)
refinement as implication (Lamport, . . .)	⋮
⋮	

Assertional and behavioural refinement 3

In a coalgebraic setting:

- The black-box view on state spaces is fundamental: “safe” states need not be constructed explicitly via suitable quotients, but are identified via invariants.

  This is in line with the standard use of class invariants in logics for object-oriented languages.

- Both assertional and behavioural refinement can be described naturally.

We start with assertional refinement.

Consider abstract and concrete coalgebraic specifications $(\mathcal{C}, E_{\mathcal{C}}, I_{\mathcal{C}})$ and $(\mathcal{A}, E_{\mathcal{A}}, I_{\mathcal{A}})$, for polynomial functors $\mathcal{C}$ and $\mathcal{A}$.

Definition (simple)

An assertional refinement consists of a natural transformation

$$\tau: \mathcal{C} \Longrightarrow \mathcal{A}$$

such that for every coalgebra $c: X \to \mathcal{C}(X)$ and initial state $x_0 \in X$ with

$$c \models E_{\mathcal{C}} \qquad \text{and} \qquad c, x_0 \models I_{\mathcal{C}}$$

there is a reachable state $x_0' \in \Diamond(\{x_0\}) \subseteq X$ such that

$$c' \models E_{\mathcal{A}} \qquad \text{and} \qquad c', x_0' \models I_{\mathcal{A}}$$

where $c' = \tau_X \circ c: X \to \mathcal{A}(X)$ is the translation of the coalgebra c via τ.

This expresses the main idea. However, it will require several "refinements". But first an example.

Earlier we specified negation-with-delay via an auxiliary method. One can also do without:

$$
\begin{array}{l}
\textbf{BEGIN}\ \mathit{DelayNeg} : \textbf{CLASSSPEC} \\
\quad \textbf{METHOD} \\
\qquad \mathsf{val}\colon \mathsf{Self} \longrightarrow 2 \\
\qquad \mathsf{neg}\colon \mathsf{Self} \times 2 \longrightarrow \mathsf{Self} \\
\quad \textbf{ASSERTION} \\
\qquad \forall b_1, b_2 \in 2.\ \mathsf{val}(\mathsf{neg}(\mathsf{neg}(x, b_1), b_2)) = \neg b_1 \\
\quad \textbf{CREATION} \\
\qquad \mathsf{val}(\mathsf{new}) = 0 \\
\qquad \forall b \in 2.\ \mathsf{val}(\mathsf{neg}(\mathsf{new}, b)) = 0 \\
\textbf{END}\ \mathit{DelayNeg}
\end{array}
$$

There is an obvious refinement of this *DelayNeg* by *DelayAuxNeg*, via the natural transformation:

$$2 \times \mathsf{Self}^2 \times 2 \xRightarrow{\langle \pi_1, \pi_2 \rangle} 2 \times \mathsf{Self}^2$$

which projects away the aux method. The same initial state can be used.

Multiple initial states

We shall now consider several modifications on the definition of assertional refinement.

The definition of assertional refinement talks about a single intial state x_0, described syntactically as $\mathsf{new}\colon \mathsf{Self}$.

However, in practice one often needs multiple initial states $(\mathsf{new}_j\colon \mathsf{Self})_{j \in J}$, for example as a function $\mathsf{new}\colon J \to \mathsf{Self}$.

This extension does not fundamentally change the picture: one now requires the existence of multiple initial states, each reachable from the given (concrete) initial states.

Thus, for convenience, we continue to assume a single initial state.

We required a natural transformation to translate form the concrete to the abstract interface. This is too restrictive:

BEGIN *Quad* : **CLASSSPEC**
METHOD
out: Self $\longrightarrow \mathbb{N}$
quad: Self $\longrightarrow$ Self
ASSERTION
$\mathsf{out}(\mathsf{quad}(x)) = 4 * \mathsf{out}(x)$
CREATION
$\mathsf{out}(\mathsf{new}) = 1$
END *Quad*

BEGIN *Dupl* : **CLASSSPEC**
METHOD
out: Self $\longrightarrow \mathbb{N}$
dupl: Self $\longrightarrow$ Self
ASSERTION
$\mathsf{out}(\mathsf{dupl}(x)) = 2 * \mathsf{out}(x)$
CREATION
$\mathsf{out}(\mathsf{new}) = 1$
END *Dupl*

It is obvious how to define an assertional refinement here: the method dupl should be used twice to obtain the behaviour of quad.

Both the interface functors of these specifications are $\mathbb{N} \times \mathsf{Self}$. But there is no suitable natural transformation

$$\mathbb{N} \times \mathsf{Self} \Longrightarrow \mathbb{N} \times \mathsf{Self}$$

When we look at the coalgebras involved we can define a suitable translation, namely:

$$\begin{array}{ccc}
(\mathsf{Self} \longrightarrow \mathbb{N} \times \mathsf{Self}) & \longrightarrow & (\mathsf{Self} \longrightarrow \mathbb{N} \times \mathsf{Self}) \\
\langle \mathsf{out}, \mathsf{dupl} \rangle & \longmapsto & \langle \mathsf{out}, \mathsf{dupl} \circ \mathsf{dupl} \rangle
\end{array}$$

We find it convenient to combine the translation of the coalgebra with the translation of the initial state into a single translation function of the form:

$$\begin{array}{ccc}
(\mathsf{Self} \longrightarrow \mathbb{N} \times \mathsf{Self}) \times \mathsf{Self} & \longrightarrow & (\mathsf{Self} \longrightarrow \mathbb{N} \times \mathsf{Self}) \times \mathsf{Self} \\
(\langle \mathsf{out}, \mathsf{dupl} \rangle, \mathsf{new}) & \longmapsto & (\langle \mathsf{out}, \mathsf{dupl} \circ \mathsf{dupl} \rangle, \mathsf{new})
\end{array}$$

We have seen the assertional refinement of *DelayNeg* by *DelayAuxNeg* which projects away the aux method.

In the other direction, we can try to define $\mathsf{aux}\colon \mathsf{Self} \to 2$ as:

$$\mathsf{aux}(x) \quad = \quad \mathsf{val}(\mathsf{neg}(x, 0))$$

But then the *DelayAuxNeg* assertions do not hold.

However, these assertions are implied by the predicate:

$$P(x) \iff \mathsf{val}(\mathsf{neg}(x, 0)) = \mathsf{val}(\mathsf{neg}(x, 1))$$

which turns out to be an invariant: it singles out the “good” states. This is standard practice in system theory and object-oriented programming.

We thus wish in assertional refinement: the abstract assertions $E_{\mathcal{A}}$ need only hold on an invariant (of the translated coalgebra):

$$\Box(E_{\mathcal{A}})(x_0') \qquad \text{for some} \qquad x_0' \in \Diamond(\{x_0\}) \cap I_{\mathcal{A}}$$

One step further is to translate the concrete coalgebra into an abstract coalgebra which is only defined on an invariant. In certain situations this is even required:

BEGIN *Abs* : **CLASSSPEC**
METHOD
a: Self $\longrightarrow \{-1, 1\}$
...
END *Abs*

BEGIN *Conc* : **CLASSSPEC**
METHOD
b: Self $\longrightarrow \mathbb{R}$
...
END *Conc*

Suppose the translation is:

$$a(x) = \frac{b(x)}{|b(x)|}$$

where we have an invariant $b(x) \neq 0$.

Clearly, this invariant is needed in order to have a well-defined translation of coalgebras.

The translation map must thus be modified into:

$$\coprod_{c:X \xrightarrow{E_{\mathcal{C}}} \mathcal{C}(X)} I_{\mathcal{C}}(c) \xrightarrow{\quad \phi \quad} \coprod_{P \subseteq X} \coprod_{c':P \xrightarrow{E_{\mathcal{A}}} \mathcal{A}(P)} I_{\mathcal{A}}(c')$$

Or, more precisely, including the reachability requirement for the abstract initial state:

$$\phi \in \prod_{c:X \xrightarrow{E_{\mathcal{C}}} \mathcal{C}(X)} \prod_{x_0 \in I_{\mathcal{C}}(c)} \coprod_{P \subseteq X} \coprod_{c':P \xrightarrow{E_{\mathcal{A}}} \mathcal{A}(P)} \Diamond(\{x_0\}) \cap I_{\mathcal{A}}(c')$$

Definition (advanced)

An assertional refinement consists of such a (second) map ϕ.

We turn to behavioural refinement for coalgebras.

Recall that the *DelayAuxNeg* specification had a "private" method aux. The intention is that it is not visible from the outside.

This can be achieved by projecting it away from the interface functor.

Such a projection can be described as a natural transformation between interface functors.

Definition

A coalgebraic specification with a view consists of a coalgebraic specification $(\mathcal{A}, E_{\mathcal{A}}, I_{\mathcal{A}})$ together with a natural transformation $\nu \colon \mathcal{A} \Longrightarrow \mathcal{A}'$ between polynomial functors. $\mathcal{A}'$ is called the public interface.

Definition

A behavioural model of $(\mathcal{A}, E_{\mathcal{A}}, I_{\mathcal{A}}, \mathcal{A} \stackrel{\nu}{\Longrightarrow} \mathcal{A}')$ consists of:

- a coalgebra $c: X \to \mathcal{A}'(X)$ of the public interface $\mathcal{A}'$
- an initial state $x_0 \in X$

which is publicly indistinguishable from an actual model: there is a coalgebra $d: Y \to \mathcal{A}(Y)$ with initial state $y_0 \in Y$, satisfying $d \models E_{\mathcal{A}}$ and $d, y_0 \models I_{\mathcal{A}}$, such that the initial states x_0 and y_0 are $\mathcal{A}'$-bisimilar:

$$x_0 \;\; {}_{c}\underline{\leftrightarrow}_{\nu \circ d} \;\; y_0$$

Proposition

For $d: Y \to \mathcal{A}(Y)$ in the definition one might as well use the final coalgebra satisfying the assertions $E_{\mathcal{A}}$.

(This final coalgebra has carrier $[\![\, \Box(E_{\mathcal{A}}) \,]\!]$, as a subcoalgebra of the final $\mathcal{T}$-coalgebra.)

Proposition

Consider a coalgebra specification $(\mathcal{A}, E_{\mathcal{A}}, I_{\mathcal{A}})$ as a specification with identity view $(\mathcal{A}, E_{\mathcal{A}}, I_{\mathcal{A}}, \mathcal{A} \stackrel{id}{\Longrightarrow} \mathcal{A})$. Then:

$(c\colon X \to \mathcal{A}(X), x_0 \in X)$ is a behavioural model of $(\mathcal{A}, E_{\mathcal{A}}, I_{\mathcal{A}}, \mathcal{A} \stackrel{id}{\Longrightarrow} \mathcal{A})$

if and only if

$(c\colon X \to \mathcal{A}(X), x_0 \in X)$ is a model of $(\mathcal{A}, \Box(E_{\mathcal{A}}), I_{\mathcal{A}})$

Definition

Consider "abstract" and "concrete" coalgebraic specifications with a view $A = (\mathcal{A}, E_{\mathcal{A}}, I_{\mathcal{A}}, \mathcal{A} \overset{\nu}{\Rightarrow} \mathcal{A}')$ and $C = (\mathcal{C}, E_{\mathcal{C}}, I_{\mathcal{C}}, \mathcal{C} \overset{\mu}{\Rightarrow} \mathcal{C}')$.

A behavioural refinement of A by C consists of a translation map for the public interfaces:

$$(X \to \mathcal{C}'(X)) \times X \xrightarrow{\quad\phi\quad} \coprod_{P \subseteq X} (P \longrightarrow \mathcal{A}'(P)) \times P$$

which maps a behavioural C-model to a behavioural A-model.

We now wish to show that the *DelayNeg* specification is als a behavioural refinement of *DelayAuxNeg*, with the view that projects away the aux method.

We simply translate val, neg, new from *DelayNeg* to val, neg, new from *DelayAuxNeg*.

We use the final *DelayAuxNeg* model with state space 2×2 and initial state $(0, 0)$.

What is thus needed is *DelayNeg*-bisimulation R which satisfies $R(\mathsf{new}, (0, 0))$. We can take:

$$R(x, (b_1, b_2)) \iff \mathsf{val}(x) = b_1 \;\wedge\; \forall b \in 2.\, \mathsf{val}(\mathsf{neg}(x, b)) = b_2.$$

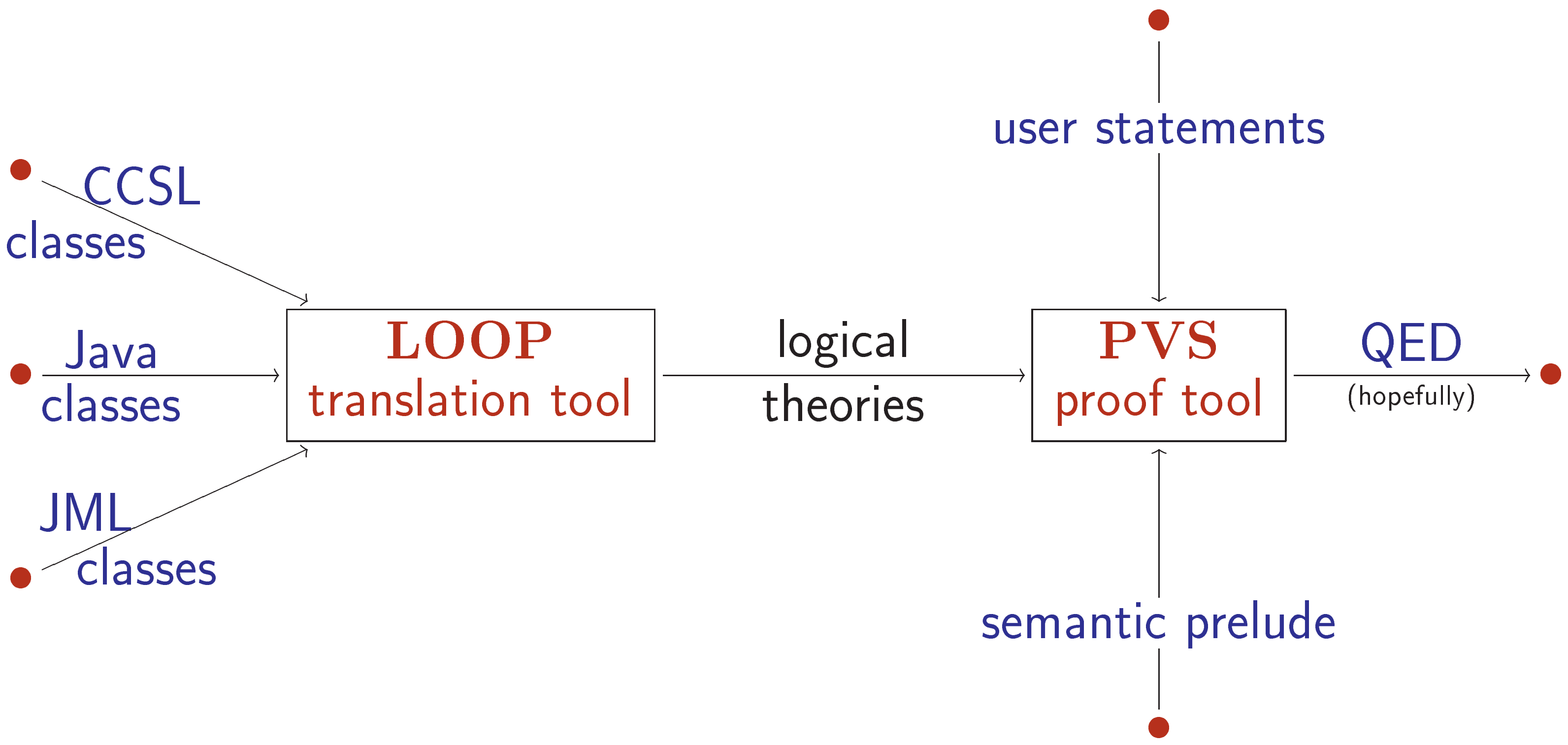
Some easy calculations show that this R does the job.

Hence, the translated initial state new is publicly indistinguishable from a *DelayAuxNeg* initial state.

Theorem

A specification $(\mathcal{C}, E_{\mathcal{C}}, I_{\mathcal{C}})$ is an assertional refinement of $(\mathcal{A}, E_{\mathcal{A}}, I_{\mathcal{A}})$ if and only if $(\mathcal{C}, E_{\mathcal{C}}, I_{\mathcal{C}}, \mathcal{C} \stackrel{id}{\Rightarrow} \mathcal{C})$ is a behavioural refinement of $(\mathcal{A}, E_{\mathcal{A}}, I_{\mathcal{A}}, \mathcal{A} \stackrel{id}{\Rightarrow} \mathcal{A})$.

The proof involves basic properties of bisimulations and invariants, and relies on an assumption that bisimilar elements satisfy the same formulas. This holds for a large class of formulas.



The LOOP tool is implemented in Ocaml and has an Emacs interface.

It is possible to use also Isabelle as a back-end theorem prover.

CCSL, for Coalgebraic Class Specification Language is an experimental specifca- tion language for classes in object-oriented languages.

It involves type-parameterisation, inheritance, aggregation (class valued attributes).

The LOOP compiler automatically generates tailor-made definitions of invariant, bisimulation, $\Box$ (henceforth) and $\Diamond$ (eventually) for the CCSL classes.

The LOOP tool provides a semantics for CCSL classes in terms of coalgebras in the logic of the back-end proof tool. (Similarly for Java and JML).

The LOOP tool is developed jointly at the universities of Nijmegen and Dresden, by:

Joachim van den Berg, Marieke Huisman, Bart Jacobs, Erik Poll (Nijmegen)

Ulrich Hensel, Hendrik Tews, Jan Rothe (Dresden).

Further topics and conclusions

- Coalgebras can form the basis for a specification formalism, with a assertion logic with bisimulations, invariants and temporal operators.
- Various subtle issues in refinement are handled quite naturally in coalgebraic specification.
- A formal language CCSL, for Coalgebraic Class Specification Language exists, together with an associated compiler. It translates classes in CCSL into logical theories of the proof tool PVS. This allows tool-assisted reasoning about class specifications, involving models, theories, and refinements. This has been used in several non-trivial examples.
- Also Java classes can be modeled as coalgebras with assertions. Smart cards with Java programs form recent important applications of Java program verification (with this semantics).

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