A Coalgebraic Approach to Bidirectional Transformations

Faris Abou-Saleh[1] James McKinna[2]

[1]Department of Computer Science, University of Oxford

[2]School of Informatics, University of Edinburgh

April 11, 2014

Overview

- Context: Unifying bidirectional transformations (bx)
- Introducing coalgebraic bx (cbx)
- Composition and behavioural equivalence
- Generalisations and Future Work

Introducing bx

- A bidirection transformation (bx) $A \Longleftrightarrow B$ describes a relationship between two entities A, B, and how changes to one result in changes to the other.
- Examples: databases and views; UML models and test suites; files in different data formats...
- Existing formalisms: relational bx; (a)symmetric/delta/edit lenses; TGGs...
- Ad-hoc definitions of composition, equivalence, ...
- Goal: A more unified perspective on these formalisms.

Previous Work

- We proposed a unifying framework (set-bx) for several of these formalisms.
- Basic idea: A, B are like cells in a store, whose values can be retrieved and modified
- Entangled state: Modifying A may affect B, and vice versa!
- Described by stateful computations

$$get_A, get_B, set_A, set_B.$$

- Natural coalgebraic perspective, inspired by the work of Power and Shkaravska on comodels for stateful computation

Introducing Coalgebraic bx

- Treat $t : A \Longleftrightarrow B$ as transition system, with state-space X, and optionally an initial state $init : X$.
  - $t.get_A : X \to A$
  - $t.get_B : X \to B$
  - $t.set_A : X \to X^A$
  - $t.set_B : X \to X^B$
- Equivalent to a coalgebra $t : X \to F_{AB}\, X$ for the functor

$$F_{AB}\, X = A \times B \times X^A \times X^B$$

Equivalent to a Moore automaton

$$F\, X = (A \times B) \times X^{A+B}$$

with constraints...

Introducing Coalgebraic bx

- Equations:

$$\begin{array}{ll} (\mathrm{CGetSet}_A) & t.set_A\ x\ (t.get_A\ x) = x \\ (\mathrm{CSetGet}_A) & t.get_A\ (t.set_A\ x\ a) = a \end{array}$$

  and similarly for B

- A coalgebraic bx $t : A \Longleftrightarrow_X B$ (with state-space X) is an F_{AB}-coalgebra satisfying the above laws.
- "Pointed", if it has an initial state $t.init : X$

A Toy Example

- $A = B = \mathbb{N}$
- bx α has state $X = A \times B$
- Operations:

$$
\begin{aligned}
\alpha.get_A\ (a,b) &= a \\
\alpha.get_B\ (a,b) &= b \\
\alpha.set_A\ a'\ (a,b) &= (a,b) \qquad \textbf{if } (a = a') \\
&= (a', b+1)\ \textit{otherwise} \\
\alpha.set_B\ b'\ (a,b) &= \ldots
\end{aligned}
$$

Advantages of a Coalgebraic Perspective

- Want to reason about, and compare, behaviour of bx.
- E.g. bx composition: given

$$\alpha : A \Longleftrightarrow_X B$$
$$\beta : B \Longleftrightarrow_Y C$$

can we define

$$\gamma = \beta \circ \alpha : A \Longleftrightarrow_Z C ?$$

Composing Coalgebraic bx

- Consider pairs of states $(x, y) : X \times Y$ that are 'B-consistent':

$$E_{\beta\alpha} = \{(x, y) : \alpha.get_B\, x = \beta.get_B\, y\}$$

- Get operations:

$$\gamma.get_A\,(x, y) = \alpha.get_A\,(x)$$
$$\gamma.get_C\,(x, y) = \beta.get_C\,(y)$$

Composing Coalgebraic bx

- Setting A and C:

$$\begin{aligned}
\gamma.set_A\ (x, y)\ a = \textbf{let}\ & x' = \alpha.set_A\ x\ a \\
& b' = \alpha.get_B\ x' \\
& y' = \beta.set_B\ y\ b' \\
\textbf{in}\ & (x', y') \\
\gamma.set_C\ (x, y)\ c = & \ldots
\end{aligned}$$

Well-behavedness of Composition

- Proposition: γ is well-defined and forms a coalgebraic bx.
- Well-behaved? I.e. Associative? Has identities?...
- Key result: Coalgebraic bx composition $\circ$ is well-behaved up to (pointed) coalgebraic bisimulation.
- Subsumes existing notions of equivalence for bx.

Pointed Coalgebraic Bisimilarity in Set

- A pointed coalgebraic bisimulation between two cbx $t_1 : A \iff_X B$ and $t_2 : A \iff_Y B$ consists of a relation $R \subseteq X \times Y$ such that:
- $(t_1.init, t_2.init) \in R$; and
- $(x, y) \in R$ implies that $t_1.get_A\ x = t_2.get_A\ y$ and for all $a : A$,

$$(t_1.set_A\ x\ a, t_2.set_A\ y\ a) \in R$$

- Similarly for B

Well-behavedness of Composition

- Theorem: Define $\alpha \equiv \alpha'$ if there is a pointed coalgebraic bisimulation between α, α'. Then, up to $\equiv$:
- Composition has identities $\iota_A, \iota_B, \ldots$

$$\alpha \circ \iota_A \equiv \iota_B \circ \alpha \equiv \alpha$$

- It is associative: $\gamma \circ (\beta \circ \alpha) \equiv (\gamma \circ \beta) \circ \alpha$
- It respects equivalence: $\alpha \equiv \alpha' \implies \beta \circ \alpha \equiv \beta \circ \alpha'$

Generalisations

- Have worked in a more general setting of “effectful bx”, where set operations may introduce effects through a strong monad M (preserving weak pullbacks):

$$F^M_{AB}\, X = A \times B \times (M\, X)^{A+B}$$

- E.g. Non-determinism (powerset monad);
- E.g. Interactive I/O:

$$MX = \nu\, Y.X + O \times Y + Y^I$$

- Categorical definitions and proofs extend beyond Set.

Future Work and Investigation

- More sophisticated classes of bx
- Combinators for coalgebraic bx
- Role and structure of final coalgebra
- Categories of metric spaces, partial orders?

Future Work and Investigation

- More sophisticated classes of bx
- Combinators for coalgebraic bx
- Role and structure of final coalgebra
- Categories of metric spaces, partial orders?
- Thank you for listening!

Postscript: Categorical Stuff

Categorical Definitions

State-space of the composition $\beta \circ \gamma$ is an equalizer

$$E_{\beta\alpha} \overset{e_{\beta\alpha}}{\dashrightarrow} X \times Y \underset{\beta.get_B \circ snd}{\overset{\alpha.get_B \circ fst}{\rightrightarrows}} B$$

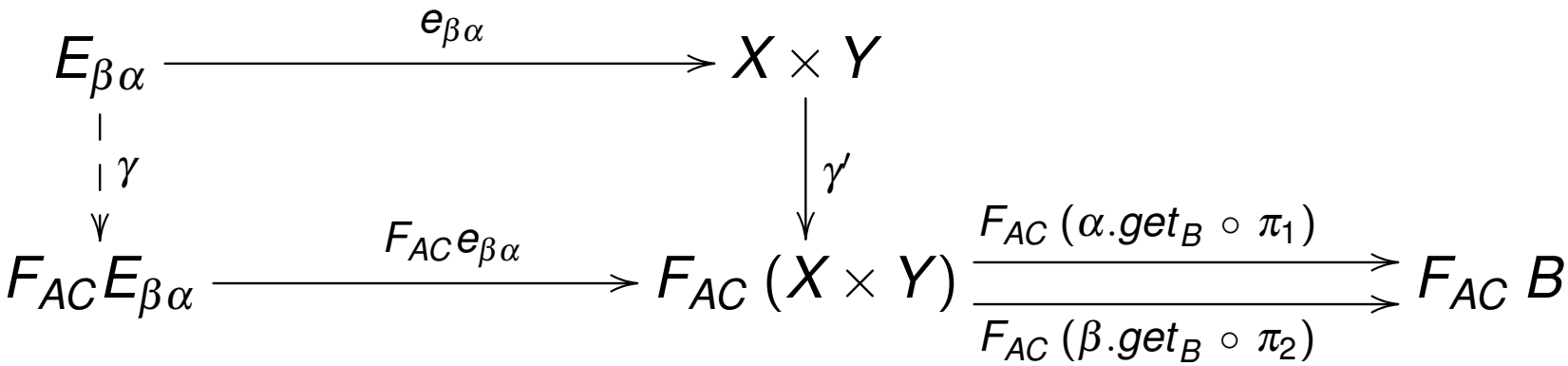
Categorical Definitions

Formally, we defined the operations of γ on pairs $X \times Y$; using the fact that they restore B-consistency (and that F_{AB} preserves (weak) pullbacks), we can "pull them back" into operations on the equalizer $E_{\beta\alpha}$.

$$
\begin{array}{ccc}
X \times Y & & \\
\downarrow{\scriptstyle \gamma'} & & \\
F_{AC}\,(X \times Y) & \underset{F_{AC}\,(\beta.get_B \circ \pi_2)}{\overset{F_{AC}\,(\alpha.get_B \circ \pi_1)}{\rightrightarrows}} & F_{AC}\,B
\end{array}
$$

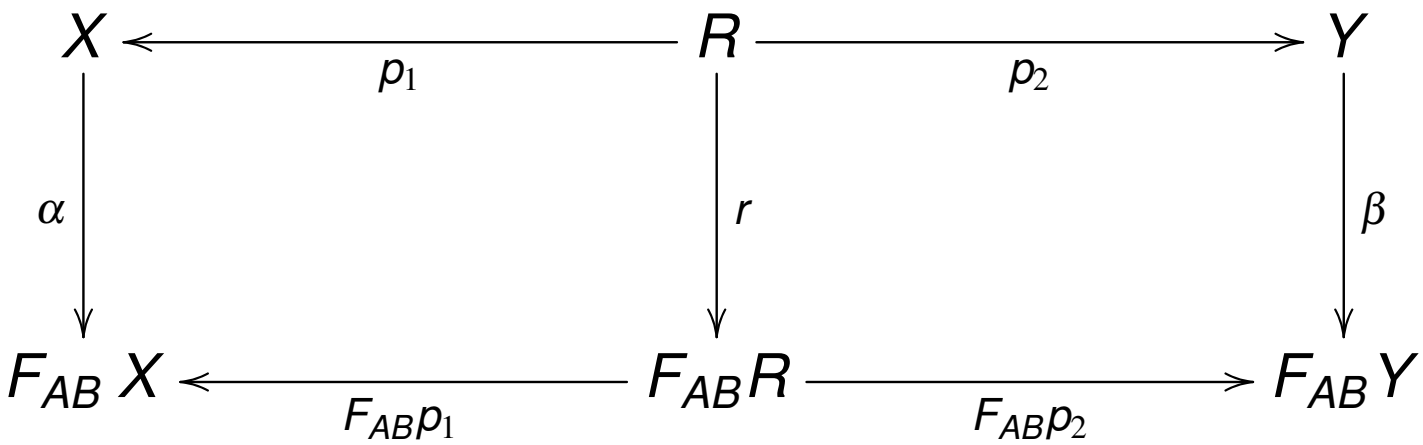
Categorical Definitions

Formally, we defined the operations of γ on pairs $X \times Y$; using the fact that they restore B-consistency (and that F_{AB} preserves (weak) pullbacks), we can "pull them back" into operations on the equalizer $E_{\beta\alpha}$.



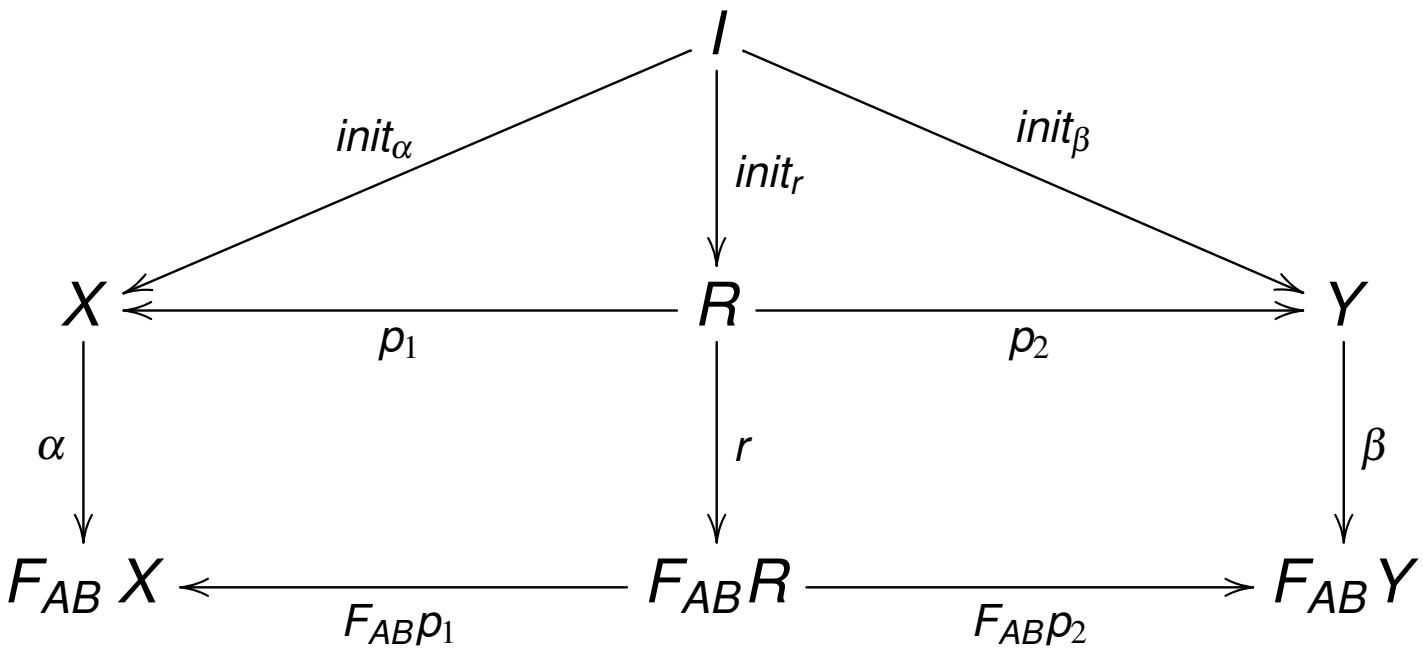
Categorical Details

Picture of a bisimulation:



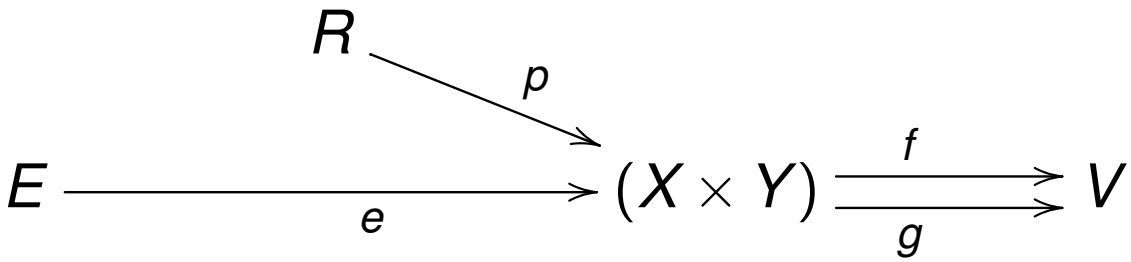
Categorical Details

Picture of a pointed bisimulation:



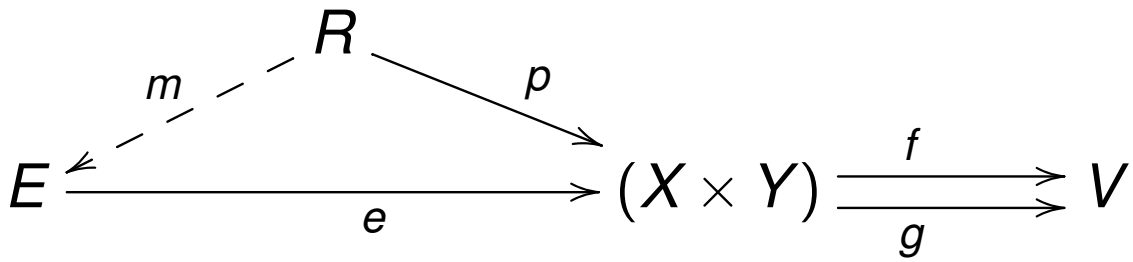
Proof Method

- How to define a bisimulation between e.g. $\alpha \circ \beta$ and $\alpha' \circ \beta$?
- Need a coalgebra $r : R \to F_{AB}\, R$, and coalgebra morphisms $r \to \alpha \circ \beta$ and $r \to \alpha' \circ \beta$
- To define $r \to \alpha \circ \beta$, first define a coalgebra morphism p, from r onto pairs $X \times Y$ (where X, Y are state-spaces of α, β)



Proof Method

- How to define a bisimulation between e.g. $\alpha \circ \beta$ and $\alpha' \circ \beta$?
- Need a coalgebra $r : R \to F_{AB}\, R$, and coalgebra morphisms $r \to \alpha \circ \beta$ and $r \to \alpha' \circ \beta$
- To define $r \to \alpha \circ \beta$, first define a coalgebra morphism p, from r onto pairs $X \times Y$ (where X, Y are state-spaces of α, β)



Simple lemma: If e and p are F-coalgebra morphisms, (and the functor $F_{..}$ weakly preserves pullbacks), then so is m.

Formalism 1: Asymmetric Lenses

- Asymmetric lens: $A \rightsquigarrow B$
- 'primary' data source A (e.g. database)
- and a secondary source B (e.g. view)
- $get : A \to B$
- $put : A \times B \to A$

$$\begin{array}{ll} \text{(GetPut)} & put\,(a, get\,(a)) = a \\ \text{(PutGet)} & get\,(put\,(a, b)) = b \end{array}$$

- Straightforward composition of lenses: $A \rightsquigarrow B$, $B \rightsquigarrow C$ gives $A \rightsquigarrow C$.

Formalism 2: Symmetric Lenses

- Symmetric lens: $A \mathrel{\dot{\longleftrightarrow}} Z\ B$
- Complement Z "data not shared between A and B".
- $putl : A \times Z \to B \times Z$
- $putr : B \times Z \to A \times Z$

$$\text{(PutrPutl)} \quad putr\ (a, z) = (b, z') \implies putl\ (b, z') = (a, z')$$

and dually (PutlPutr)

- Composition of $A \mathrel{\dot{\longleftrightarrow}} X\ B$ and $B \mathrel{\dot{\longleftrightarrow}} Y\ C$ well-defined up to equivalence.

Formalism 3: Relational bx

- Relational bx: $(R, \overrightarrow{R}, \overleftarrow{R})$
- Consistency relation $R \subseteq (A \times B)$
- $\overrightarrow{R} : A \times B \to B$
- $\overleftarrow{R} : A \times B \to A$

$$\begin{array}{lll} \text{(Correct)} & (a, \overrightarrow{R}\,(a,b)) \in R & \\ & (\overleftarrow{R}\,(a,b), b) \in R & \\ \text{(Hippocratic)} & (a,b) \in R \implies & \overrightarrow{R}\,(a,b) = b \\ & & \overleftarrow{R}\,(a,b) = a \end{array}$$

- 'Composition' exists, but is no longer a relational bx...

Example Application: Database of Composers

- Take a database of musical composers:

Name	Nationality	Lifespan
Bach	Germany	1685-1750
Mozart	Austria	1756-1791
Schubert	Austria	1797-1828
...	...	...

Example Application: Database of Composers

- One could take views of this database

Name	Nationality
Bach	Germany
Mozart	Austria
...	...

or

Name	Lifespan
Bach	1685-1750
Mozart	1756-1791
...	...

  and update the database whenever a view is edited;
- or if two users have different views of the same database, one could propagate changes between the views.
- In general, there are many design choices and subtleties involved in how we propagate changes.
- Diverse collection of bx formalisms: relational bx, symmetric and asymmetric lenses, delta lenses, edit lenses...