Recursion in Coalgebras

Mauro Jaskelioff
mjj@cs.nott.ac.uk

School of Computer Science & IT

FoP Away Day 2007
Outline

- Brief overview of coalgebras.
- The problem of divergence when considering unguarded recursion.
- Different approaches to solving the problem.
The coalgebraic Approach
A Quick Overview

- Coalgebras are the dual of algebras
The coalgebraic Approach
A Quick Overview

- Coalgebras are the dual of algebras
- Coalgebras provide elegant models for dynamic systems
The coalgebraic Approach
A Quick Overview

- Coalgebras are the dual of algebras
- Coalgebras provide elegant models for dynamic systems
  - automatas

Coalgebras are defined over a behaviour functor $B$

$B$ determines what is observable in the system.

The carrier $X$ can be thought of as a set of states.
The coalgebraic Approach
A Quick Overview

- Coalgebras are the dual of algebras
- Coalgebras provide elegant models for dynamic systems
  - automatas
  - transition systems
The coalgebraic Approach
A Quick Overview

- Coalgebras are the dual of algebras
- Coalgebras provide elegant models for dynamic systems
  - automatas
  - transition systems
  - abstract machines
The coalgebraic Approach
A Quick Overview

- Coalgebras are the dual of algebras
- Coalgebras provide elegant models for dynamic systems
  - automatas
  - transition systems
  - abstract machines
  - object oriented systems
Coalgebras are the dual of algebras
Coalgebras provide elegant models for dynamic systems
- automatas
- transition systems
- abstract machines
- object oriented systems
Coalgebras are defined over a *behaviour functor* $B$
The coalgebraic Approach
A Quick Overview

- Coalgebras are the dual of algebras
- Coalgebras provide elegant models for dynamic systems
  - automatas
  - transition systems
  - abstract machines
  - object oriented systems
- Coalgebras are defined over a *behaviour functor B*
- $B$ determines what is observable in the system.
The coalgebraic Approach
A Quick Overview

- Coalgebras are the dual of algebras
- Coalgebras provide elegant models for dynamic systems
  - automatas
  - transition systems
  - abstract machines
  - object oriented systems
- Coalgebras are defined over a behaviour functor $B$
- $B$ determines what is observable in the system.
- More concretely: A coalgebra is an arrow
  $$X \rightarrow BX$$

The carrier $X$ can be thought of as a set of states.
A Simple Coalgebra: LTS

Labelled transition systems are typical examples of coalgebras. The behaviour in this case is the Set functor

\[ BX = \mathcal{P}(A \times X) \]
A Simple Coalgebra: LTS

Labelled transition systems are typical examples of coalgebras. The behaviour in this case is the Set functor

\[ BX = \mathcal{P}(A \times X) \]

As an example, consider the set of states \( X = \{x, y, z\} \), and set of actions \( A = \{a, b, c, d\} \)

The system

The system

\[
\begin{align*}
    d & \rightarrow x \\
    a & \rightarrow y \\
    b & \rightarrow z \\
    c & \leftarrow y
\end{align*}
\]
A Simple Coalgebra: LTS

Labelled transition systems are typical examples of coalgebras. The behaviour in this case is the $\text{Set}$ functor

$$BX = \mathcal{P}(A \times X)$$

As an example, consider the set of states $X = \{x, y, z\}$, and set of actions $A = \{a, b, c, d\}$.

The system

\[
\begin{array}{ccc}
  & x & \\
  d & \downarrow & b \\
  y & \leftarrow & z \\
a & \downarrow & c
  \\
\end{array}
\]

is given by the following coalgebra

\[
\alpha : X \rightarrow \mathcal{P}(A \times X)
\]

\[
\alpha(x) = \{(a, y), (b, z)\}
\]

\[
\alpha(y) = \{(d, x)\}
\]

\[
\alpha(z) = \{(c, y)\}
\]
Complete Behaviour

- A coalgebra $\alpha : X \rightarrow BX$ yields one “step” of behaviour.
A coalgebra $\alpha : X \to BX$ yields one “step” of behaviour. The complete abstract behaviour of a system is obtained by finality.

\[
\begin{align*}
X &\twoheadrightarrow !_{\alpha} \ni X.BX \\
\alpha &\downarrow \\
BX &\cong B(\nu X.BX)
\end{align*}
\]
A coalgebra $\alpha : X \to BX$ yields one “step” of behaviour.

The complete abstract behaviour of a system is obtained by finality.

The unique map $!_\alpha$ into the final coalgebra is often called *unfold*.
Observational equivalence

The canonical notion of observational equivalence is Coalgebraic $B$-bisimulation
Observational equivalence

The canonical notion of observational equivalence is

Coalgebraic $B$-bisimulation

For $s \in S$, $t \in T$, $R \subseteq S \times T$

$$\langle s, \alpha \rangle \sim_B \langle t, \beta \rangle \iff \exists \gamma$$
Observational equivalence

The canonical notion of observational equivalence is

Coalgebraic $B$-bisimulation

For $s \in S$, $t \in T$, $R \subseteq S \times T$

$$\langle s, \alpha \rangle \sim_B \langle t, \beta \rangle \iff \exists \gamma$$
Example: Bisimulation for LTS

For the case of labelled transition systems, the previous diagram means \((s, t) \in R\) iff

\[\forall (a, s') \in \alpha(s). \ \exists (a, t') \in \beta(t) \land (s', t') \in R\]

\[\forall (a, t') \in \beta(t). \ \exists (a, s') \in \alpha(s) \land (s', t') \in R\]

\[\alpha(s) = \emptyset \iff \beta(t) = \emptyset\]

which corresponds which the ordinary notion of bisimulation.
A model of Recursion

- Terms of a language as carrier of a coalgebra (which defines the semantics of the language).

- We'll model recursion by systems of equations

\[
\psi(x) = a; x_0; \psi(b; x_1)
\]

When are equations guarded?

- Syntactically guarded
  - RHS must begin with a non-recursive operator.
  - Avoids silly equations like \(x = x\) or cycles \(x = y, y = x\), etc.

- Behaviourally guarded
  - It's possible to extract behaviour from the RHS.
  - \(\phi\) is syntactically but not behaviourally guarded.
A model of Recursion

- Terms of a language as carrier of a coalgebra (which defines the semantics of the language).
- We’ll model recursion by systems of equations

Example

\[ \psi(x) = a; x; \psi(b;x) \]

When are equations guarded?

- Syntactically guarded
  - RHS must begin with a non-recursive operator.
  - Avoids silly equations like \( x = x \) or cycles \( x = y, y = x \), etc.

- Behaviourally guarded
  - It’s possible to extract behaviour from the RHS.
  - \( \phi \) is syntactically but not behaviourally guarded
A model of Recursion

- Terms of a language as carrier of a coalgebra (which defines the semantics of the language).
- We’ll model recursion by systems of equations
- Example

\[
\psi(x) = a ; x ; \psi(b ; x)
\]

\[
\varphi = \varphi ; \psi(a)
\]
A model of Recursion

- Terms of a language as carrier of a coalgebra (which defines the semantics of the language).
- We’ll model recursion by systems of equations
- Example

\[ \psi(x) = a \cdot x \cdot \psi(b; x) \]
\[ \varphi = \varphi \cdot \psi(a) \]
A model of Recursion

- Terms of a language as carrier of a coalgebra (which defines the semantics of the language).
- We’ll model recursion by systems of equations
- Example

\[
\psi(x) = a ; x ; \psi(b; x) \\
\varphi = \varphi ; \psi(a)
\]

When are equations guarded?
A model of Recursion

- Terms of a language as carrier of a coalgebra (which defines the semantics of the language).
- We’ll model recursion by systems of equations
- Example

\[
\psi(x) = a \cdot x \cdot \psi(b; x)
\]
\[
\phi = \phi \cdot \psi(a)
\]

When are equations guarded?
- Syntactically guarded
  - RHS must begin with a non-recursive operator.
  - Avoids silly equations like \(x = x\) or cycles \(x = y, y = x\), etc.
- behaviourally guarded.
  - It’s possible to extract behaviour from the RHS.
  - \(\phi\) is syntactically but not behaviourally guarded
A model of Recursion

- Terms of a language as carrier of a coalgebra (which defines the semantics of the language).
- We’ll model recursion by systems of equations
- Example

\[
\psi(x) = a \cdot x \cdot \psi(b; x) \\
\varphi = \varphi \cdot \psi(a)
\]

When are equations guarded?
- Syntactically guarded
  - RHS must begin with a non-recursive operator.
A model of Recursion

- Terms of a language as carrier of a coalgebra (which defines the semantics of the language).
- We’ll model recursion by systems of equations
- Example

\[
\begin{align*}
\psi(x) & = a ; x ; \psi(b; x) \\
\phi & = \phi ; \psi(a)
\end{align*}
\]

When are equations guarded?
- Syntactically guarded
  - RHS must begin with a non-recursive operator.
  - Avoids silly equations like \( x = x \) or cycles \( x = y, y = x \), etc.
A model of Recursion

- Terms of a language as carrier of a coalgebra (which defines the semantics of the language).
- We’ll model recursion by systems of equations
- Example

\[
\psi(x) = a \cdot x \cdot \psi(b; x) \\
\phi = \phi \cdot \psi(a)
\]

When are equations guarded?
- Syntactically guarded
  - RHS must begin with a non-recursive operator.
  - Avoids silly equations like \( x = x \) or cycles \( x = y, y = x \), etc.
- Behaviourally guarded.
A model of Recursion

- Terms of a language as carrier of a coalgebra (which defines the semantics of the language).
- We’ll model recursion by systems of equations
- Example

\[ \psi(x) = a ; x ; \psi(b ; x) \]
\[ \phi = \phi ; \psi(a) \]

When are equations guarded?
- Syntactically guarded
  - RHS must begin with a non-recursive operator.
  - Avoids silly equations like \( x = x \) or cycles \( x = y, y = x \), etc.
- Behaviourally guarded.
  - It’s possible to extract behaviour from the RHS.
A model of Recursion

- Terms of a language as carrier of a coalgebra (which defines the semantics of the language).
- We’ll model recursion by systems of equations.
- Example

\[
\psi(x) = a ; x ; \psi(b ; x) \\
\varphi = \varphi ; \psi(a)
\]

When are equations guarded?
- Syntactically guarded
  - RHS must begin with a non-recursive operator.
  - Avoids silly equations like \(x = x\) or cycles \(x = y, y = x\), etc.
- Behaviourally guarded.
  - It’s possible to extract behaviour from the RHS.
  - \(\varphi\) is syntactically but not behaviourally guarded.
The Problem with Unguarded Equations

- Behaviourally guarded equations are not problematic: one can always obtain a coalgebra for them.
The Problem with Unguarded Equations

- Behaviourally guarded equations are not problematic: one can always obtain a coalgebra for them.

\[\psi(x) \mapsto \begin{cases} (a, x; \psi(b; x)) \\ \text{new state} \end{cases}\]
The Problem with Unguarded Equations

- Behaviourally guarded equations are not problematic: one can always obtain a coalgebra for them.

\[ \psi(x) \mapsto \begin{cases} (a, x; \psi(b; x)) \end{cases} \]

- If we cannot obtain behaviour from the RHS of the equation, then the only possible behaviour is divergence.

\[ \varphi \mapsto ??? \]
The Problem with Unguarded Equations

- Behaviourally guarded equations are not problematic: one can always obtain a coalgebra for them.

\[
\psi(x) \mapsto \begin{cases} 
(a, x; \psi(b; x)) \\
\text{new state}
\end{cases}
\]

- If we cannot obtain behaviour from the RHS of the equation, then the only possible behaviour is divergence.

\[
\varphi \mapsto ???
\]

- How to express divergence coalgebraically?
1) Recursion as Syntactic sugar

- The symbols defined by equations are not part of the language. They are syntactic sugar for their infinite expansions.
1) Recursion as Syntactic sugar

- The symbols defined by equations are not part of the language. They are syntactic sugar for their infinite expansions.

- Programs can be infinite.
1) Recursion as Syntactic sugar

- The symbols defined by equations are not part of the language. They are syntactic sugar for their infinite expansions.

- Programs can be infinite.

- This approach needs a category with more structure like CPO.
1) Recursion as Syntactic sugar

- The symbols defined by equations are not part of the language. They are syntactic sugar for their infinite expansions.

- Programs can be infinite.

- This approach needs a category with more structure like CPO.

1) Recursion as Syntactic sugar

- The symbols defined by equations are not part of the language. They are syntactic sugar for their infinite expansions.

- Programs can be infinite.

- This approach needs a category with more structure like CPO.


- It’s a domain-theory-oriented solution.
2) Adding divergence to the behaviour

- Consider the behaviour $B + 1$, where we denote the element of 1 by $\bot$. 

Drawback: A coalgebra may detect divergence.

naughty $(t) \mapsto \begin{cases} \text{stop} & \text{if } \alpha(t) = \bot \\ \bot & \text{else} \end{cases}$

If we work in the category $\text{Set}$, this might be acceptable!
2) Adding divergence to the behaviour

- Consider the behaviour $B + 1$, where we denote the element of 1 by $\bot$.

- We can then define $\varphi \mapsto \bot$. 

  Drawback: A coalgebra may detect divergence.

  naughty $(t) \mapsto$ if $\alpha(t) = \bot$ then stop else $\bot$ 

  If we work in the category $\text{Set}$, this might be acceptable!
2) Adding divergence to the behaviour

- Consider the behaviour $B + 1$, where we denote the element of 1 by $\perp$.

- We can then define $\varphi \mapsto \perp$.

- Drawback: A coalgebra may detect divergence.
2) Adding divergence to the behaviour

- Consider the behaviour $B + 1$, where we denote the element of 1 by $⊥$.
- We can then define $ϕ ↦ → ⊥$.
- Drawback: A coalgebra may detect divergence.
- \(naughty(t) \mapsto \) if $α(t) = ⊥$ then \(stop\) else $⊥$
2) Adding divergence to the behaviour

- Consider the behaviour \( B + 1 \), where we denote the element of 1 by \( \perp \).

- We can then define \( \varphi \mapsto \perp \).

- Drawback: A coalgebra may detect divergence.

- \textit{naughty}(t) \mapsto \text{if } \alpha(t) = \perp \text{ then } \text{stop} \text{ else } \perp

- If we work in the category \( \text{Set} \), this might be acceptable!
3) Ignoring expansions

- Consider a behaviour $B\perp X = X + BX$
3) Ignoring expansions

- Consider a behaviour $B_{\perp}X = X + BX$

- But equation expansions are visible!
3) Ignoring expansions

- Consider a behaviour $B \perp X = X + BX$
- But equation expansions are visible!
- Given an equation $\chi = a$,

$$\chi \not\approx a$$
3) Ignoring expansions

- Consider a behaviour $B \downarrow X = X + BX$
- But equation expansions are visible!
- Given an equation $\chi = a$,

$$\chi \not\sim a$$

- We need to consider a notion of observation that ignores equation expansion.
3) Transforming the coalgebra

- We define an endofunctor of $B\perp$-coalgebras

\[
\Phi_n : B\perp\text{-Coalg} \to B\perp\text{-Coalg}
\]
\[
\Phi_0(k) = X \xrightarrow{k} X + BX
\]
\[
\Phi_{n+1}(k) = X \xrightarrow{\Phi_n(k)} X + BX \xrightarrow{[k, id]} X + BX
\]
3) Transforming the coalgebra

- We define an endofunctor of $B_\bot$-coalgebras

$$\Phi_n : B_\bot\text{-Coalg} \to B_\bot\text{-Coalg}$$

$$\Phi_0(k) = X \xrightarrow{k} X + BX$$

$$\Phi_{n+1}(k) = X \xrightarrow{\Phi_n(k)} X + BX \xrightarrow{[k, id]} X + BX$$

- Given $\alpha, \beta : B_\bot\text{-Coalg}$. We define

$$\langle s, \alpha \rangle \approx_B^n \langle t, \beta \rangle$$

to be

$$\langle s, \Phi_n(\alpha) \rangle \sim_B \langle t, \Phi_n(\beta) \rangle$$
3) Transforming the coalgebra

- We define an endofunctor of $B_\bot$-coalgebras

$$
\Phi_n : B_\bot\text{-Coalg} \to B_\bot\text{-Coalg}
$$

$$
\Phi_0(k) = X \xrightarrow{k} X + BX
$$

$$
\Phi_{n+1}(k) = X \xrightarrow{\Phi_n(k)} X + BX \xrightarrow{[k, id]} X + BX
$$

- Given $\alpha, \beta : B_\bot\text{-Coalg}$. We define

$$
\langle s, \alpha \rangle \approx^n_B \langle t, \beta \rangle
$$

to be

$$
\langle s, \Phi_n(\alpha) \rangle \sim_B \langle t, \Phi_n(\beta) \rangle
$$

- Claim: if we have $n$ equations, considering $\Phi_n$ is enough to eliminate all finite sequences of expansions.
Summary

- Coalgebras provide a nice model of dynamic systems, but
Summary

- Coalgebras provide a nice model of dynamic systems, but divergence can be problematic to model coalgebraically.
Summary

- Coalgebras provide a nice model of dynamic systems, but divergence can be problematic to model coalgebraically.
- We can transform a coalgebra so that it ignores a given number of silent steps.
Summary

- Coalgebras provide a nice model of dynamic systems, but
- Divergence can be problematic to model coalgebraically.
- We can transform a coalgebra so that it ignores a given number of silent steps.
Summary

- Coalgebras provide a nice model of dynamic systems, but divergence can be problematic to model coalgebraically.
- We can transform a coalgebra so that it ignores a given number of silent steps.

Future Work

- Remove dependence from $n$ by some $\Phi_\omega$.
- Correspondence between $\approx_{B\perp}$ and what’s expected in concrete cases.