Coalgebraic Structures in Program Construction

Luís S. Barbosa

DI - UNIV. MINHO

lsb@di.uminho.pt

UNU-IIST

September 2002

Motivation

How to drive software production into solid engineering standards?

Motivation

How to drive software production into solid engineering standards?

An old recipe from school physics:

- Understand the problem
- Create a **mathematical model**
- **Reason** within the model
- Calculate a **solution** ($\equiv$ an **implementation**)

The Approach

- combine previous experience with model oriented formal design methods (VDM and CAMILA [AMAST'97])
- ... with the 'tradition' of program calculi driven by (initial or final) type specifications
  - *cf.*, systematic derivation of algorithms in a way that correctness is guaranteed by construction
  - 'twin' concern in functional programming [McCarthy original papers, Backus FP]
  - Bird-Meertens 'school' [BM87] of mathematics of program construction

The Approach

Can this emphasis on calculation and the underlying modelling principles be extended to the world of dynamic, state-based systems?

The Approach

Can this emphasis on calculation and the underlying modelling principles be extended to the world of dynamic, state-based systems?

Key: resort to the algebra *vs* coalgebra duality as a mathematical 'explanation' of the intuitive symmetry between data and behavioural structures

The Approach

Can this emphasis on calculation and the underlying modelling principles be extended to the world of dynamic, state-based systems?

Key: resort to the algebra *vs* coalgebra duality as a mathematical 'explanation' of the intuitive symmetry between data and behavioural structures

- ... a 'big' topic: from 'non well-founded' sets [Aczel88]
- to 'behavioural satisfaction' [Reichel81] and 'final semantics' [Plotkin, Rutten, Turi 93]
- to axiomatic 'coalgebraic specification [Jacobs97] and 'coalgebraic logic' [Moss99]
- ... partly documented in the CMCS series (from 1998)

The Basic Symmetry

(Initial) algebras are abstract description of data structures

$$[\mathsf{nil}, \mathsf{cons}] : \mathbf{1} + O \times L \longrightarrow L$$

In general:

a tool box: 

an assembly process:  artifact $\xrightarrow{d}$ artifact

The emphasis is on construction

The Basic Symmetry

(Final) coalgebras are abstract descriptions of systems' behaviours

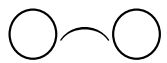
$$\langle \mathsf{at}, \mathsf{m} \rangle : U \longrightarrow O \times U$$

The Basic Symmetry

(Final) coalgebras are abstract descriptions of systems' behaviours

$$\langle \mathsf{at}, \mathsf{m} \rangle : U \longrightarrow O \times U$$

In general:

a lens: 

an observation structure: universe $\xrightarrow{f}$ universe

The emphasis is on observation

The Observational Viewpoint

Dynamical Systems

- internal state space (‘memory’ and persistence)
- possibility of interaction with other components during the overall computation
- observable through well-defined interfaces to ensure flow of data

found everywhere: from sophisticated plant control systems to formal automata or domestic appliances.

The Observational Viewpoint

Observation Shapes

- ‘opaque’

- black & white

- colouring

... in each case the colour set acts as a classifier of the state space

The Observational Viewpoint

Observation Shapes

- partiality

$$\bigcirc\!\!\frown\!\!\bigcirc\, U \;=\; U + \mathbf{1}$$

The Observational Viewpoint

Observation Shapes

- partiality

$$\bigcirc\!\!\frown\!\!\bigcirc\, U \;=\; U + \mathbf{1}$$

- visible attributes

$$\bigcirc\!\!\frown\!\!\bigcirc\, U \;=\; O \times U$$

The Observational Viewpoint

Observation Shapes

- partiality

$$\bigcirc\!\!\frown\!\!\bigcirc\, U \;=\; U + \mathbf{1}$$

- visible attributes

$$\bigcirc\!\!\frown\!\!\bigcirc\, U \;=\; O \times U$$

- external stimulus

$$\bigcirc\!\!\frown\!\!\bigcirc\, U \;=\; U^{I}$$

The Observational Viewpoint

Observation Shapes

- partiality

$$\bigcirc\!\!\frown\!\!\bigcirc\, U \;=\; U + \mathbf{1}$$

- visible attributes

$$\bigcirc\!\!\frown\!\!\bigcirc\, U \;=\; O \times U$$

- external stimulus

$$\bigcirc\!\!\frown\!\!\bigcirc\, U \;=\; U^{I}$$

- non determinism

$$\bigcirc\!\!\frown\!\!\bigcirc\, U \;=\; \mathcal{P}U$$

A parenthesis

As our underlying ‘semantic universe’ assume an elementary

space of types and typed arrows ...

As our underlying ‘semantic universe’ assume an elementary

- space of types and typed arrows ...
- with the structure of a (partial) monoid

As our underlying ‘semantic universe’ assume an elementary

- space of types and typed arrows ...
- with the structure of a (partial) monoid
- ... taken in the sequel as sets and set-theoretical functions

As our underlying ‘semantic universe’ assume an elementary

- space of types and typed arrows ...
- with the structure of a (partial) monoid
- ... taken in the sequel as sets and set-theoretical functions

Function combinators are defined by universal arrows

- associated to the product, sum and exponential constructions

As our underlying ‘semantic universe’ assume an elementary

- space of types and typed arrows ...
- with the structure of a (partial) monoid
- ... taken in the sequel as sets and set-theoretical functions

Function combinators are defined by universal arrows

- associated to the product, sum and exponential constructions
- which behave ... as they should do (ccc structure)

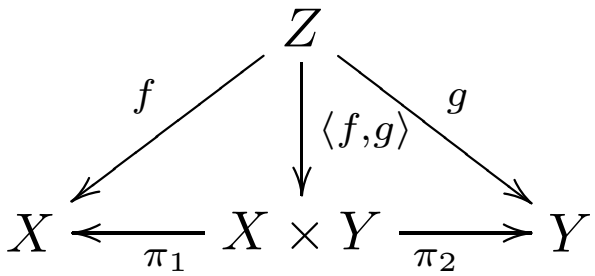
As our underlying ‘semantic universe’ assume an elementary

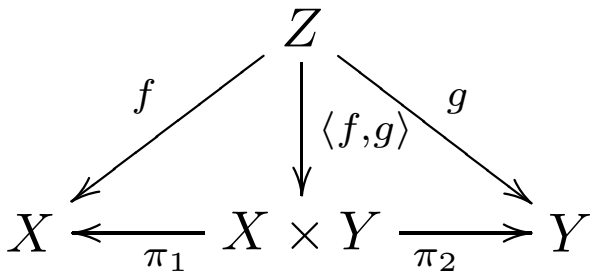
- space of types and typed arrows ...
- with the structure of a (partial) monoid
- ... taken in the sequel as sets and set-theoretical functions

Function combinators are defined by universal arrows

- associated to the product, sum and exponential constructions
- which behave ... as they should do (ccc structure)

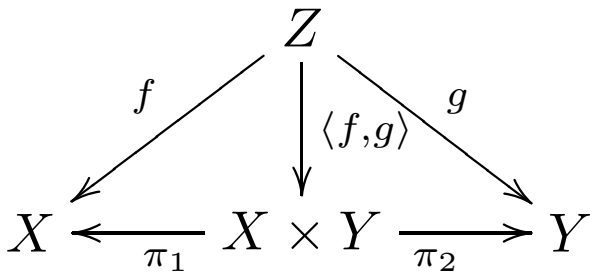
Example: pointfree presentation of the product construction


Z
f
⟨f,g⟩
g
X
π1
X × Y
π2
Y



Universal property:

$$k = \langle f, g \rangle \quad \Leftrightarrow \quad \pi_1 \cdot k = f \ \wedge \ \pi_2 \cdot k = g$$



Universal property:

$$k = \langle f, g \rangle \quad \Leftrightarrow \quad \pi_1 \cdot k = f \ \wedge \ \pi_2 \cdot k = g$$

Laws:

$$\pi_1 \cdot \langle f, g \rangle = f \ , \pi_2 \cdot \langle f, g \rangle = g$$

$$\langle \pi_1, \pi_2 \rangle \ = \ \mathsf{id}_{X \times Y}$$

$$\langle g, h \rangle \cdot f \ = \ \langle g \cdot f, h \cdot f \rangle$$

$$(i \times j) \cdot \langle g, h \rangle \ = \ \langle i \cdot g, j \cdot h \rangle$$

End of parenthesis

)

Systems & Behaviours

Simple Objects

$$p = \langle \mathsf{at}, \mathsf{m} \rangle : U \longrightarrow O \times U$$

Systems & Behaviours

Simple Objects

$$p \;=\; \langle \mathsf{at}, \mathsf{m} \rangle : U \longrightarrow O \times U$$

The behaviour of p at (from) a state u is revealed by successive observations (experiments):

$$[\![(p)]\!]\; u \;=\; <\mathsf{at}\; u,\; \mathsf{at}\; (\mathsf{m}\; u),\; \mathsf{at}\; (\mathsf{m}\; (\mathsf{m}\; u)), ...>$$

Systems & Behaviours

Mealy Machines

$$p = \overline{\langle \mathsf{at}, \mathsf{m} \rangle} : U \longrightarrow (O \times U)^I$$

Systems & Behaviours

Mealy Machines

$$p \;=\; \overline{\langle \mathsf{at}, \mathsf{m} \rangle} : U \longrightarrow (O \times U)^I$$

Behaviours are elements of O^{I^+} — *i.e.*, functions from non empty sequences of I to O

$$[\![(p)]\!]\; u \;=\; \lambda <s : i> \;.\;\; \mathsf{at}\; \langle (\mathsf{next}\; u)\; s, i \rangle$$

where

$$(\mathsf{next}\; u) <> = u \qquad \text{and} \qquad (\mathsf{next}\; u) <s : i> = \mathsf{m}\; \langle (\mathsf{next}\; u)\; s, i \rangle$$

Systems & Behaviours

Fact: The behaviours of ◯⌒◯-systems form a ◯⌒◯-system

$$\omega \;=\; \overline{\langle \mathsf{root}, \mathsf{branches} \rangle} : O^{I^+} \longrightarrow (O \times O^{I^+})^I$$

where,

$$\mathsf{root}\ \langle \phi, i \rangle = \phi\ {<}i{>} \qquad \text{and} \qquad \mathsf{branches}\ \langle \phi, i \rangle = \lambda\, s\ .\ \ \phi\ {<}i : s{>}$$

Systems & Behaviours

Relating Systems

A morphism is a function connecting the state spaces which preserves the dynamics, *i.e.*,

$$\begin{array}{ccc} U \times I & \xrightarrow{\langle \mathsf{at},\mathsf{m} \rangle} & O \times U \\ {\scriptstyle h \times \mathsf{id}}\downarrow & & \downarrow{\scriptstyle \mathsf{id} \times h} \\ V \times I & \xrightarrow{\langle \mathsf{at}',\mathsf{m}' \rangle} & O \times V \end{array}$$

I.e.,

$$\mathsf{at} = \mathsf{at}' \cdot (h \times \mathsf{id})$$

and

$$h \cdot \mathsf{m} = \mathsf{m}' \cdot (h \times \mathsf{id})$$

Systems & Behaviours

Fact: $[\![(p)]\!]$ is a morphism from p to ω

Systems & Behaviours

Fact: $[\![(p)]\!]$ is a morphism from p to ω

Proof (check both conditions)

$$
\begin{array}{ll}
 & (\mathsf{root} \cdot ([\![(p)]\!] \times \mathsf{id}))\ \langle u, i\rangle \\
= & \quad \{\ \mathsf{root}\ \text{defi nition}\ \} \\
 & ([\![(p)]\!]\ u)\ <i> \\
= & \quad \{\ [\![(p)]\!]\ \text{defi nition}\ \} \\
 & \mathsf{at}\ \langle (\mathsf{next}\ u)\ <>, i\rangle \\
= & \quad \{\ \mathsf{next}\ \text{defi nition}\ \} \\
 & \mathsf{at}\ \langle u, i\rangle
\end{array}
$$

Systems & Behaviours

and

$$
\begin{aligned}
& (\mathsf{branches} \cdot ([\![p]\!] \times \mathsf{id}))\ \langle u, i\rangle \\
= & \quad \{\ \mathsf{branches}\ \text{defi nition}\ \} \\
& \lambda <s : j> .\ ([\![p]\!]\ u) <i : s : j> \\
= & \quad \{\ [\![p]\!]\ \text{defi nition}\ \} \\
& \lambda <s : j> .\ \mathsf{at}\ \langle (\mathsf{next}\ u) <i : s>, j\rangle \\
= & \quad \{\ \mathsf{next}\ \text{defi nition}\ \} \\
& \lambda <s : j> .\ \mathsf{at}\ \langle (\mathsf{next\ m}\ \langle u, i\rangle)s, j\rangle \\
= & \quad \{\ [\![p]\!]\ \text{defi nition}\ \} \\
& [\![p]\!]\ (\mathsf{m}\ \langle u, i\rangle)
\end{aligned}
$$

Systems & Behaviours

Fact: Morphisms preserve behaviour

$$[\![(p)]\!]\ u \ = \ [\![(p)]\!]\ h\ u$$

Systems & Behaviours

What's special about $\langle O^{I^+}, \omega \rangle$?

- There is always a morphism — $[\![(p)]\!]$ — to it from any $\langle U, p = \overline{\langle \mathsf{at}, \mathsf{m} \rangle} \rangle$

Systems & Behaviours

What's special about $\langle O^{I^+}, \omega \rangle$?

- There is always a morphism — $[\![(p)]\!]$ — to it from any $\langle U, p = \overline{\langle \mathsf{at}, \mathsf{m} \rangle} \rangle$
- Because morphisms preserve behaviour, such a morphism is unique

Systems & Behaviours

What's special about $\langle O^{I^+}, \omega \rangle$?

- There is always a morphism — $[\![(p)]\!]$ — to it from any $\langle U, p = \overline{\langle \mathsf{at}, \mathsf{m} \rangle} \rangle$
- Because morphisms preserve behaviour, such a morphism is unique
  - $\omega = \langle \mathsf{root}, \mathsf{branches} \rangle$ is the final Mealy machine
  - example of a characterisation by a universal property

Going Generic

- Mealy machines (on I and O) were introduced as systems observed through

$$\bigcirc\!\!\frown\!\!\bigcirc \;=\; (O \times -)^I$$

$\bigcirc\!\!\frown\!\!\bigcirc$ as a mapping to decompose the ‘observable-universe’ U into an ’observation context’ $(O \times U)^I$

Going Generic

- Mealy machines (on I and O) were introduced as systems observed through

$$\bigcirc\!\!\frown\!\!\bigcirc \;=\; (O \times -)^I$$

  $\bigcirc\!\!\frown\!\!\bigcirc$ as a mapping to decompose the 'observable-universe' U into an 'observation context' $(O \times U)^I$

- The pair $\langle U, p \rangle$ is called a $\bigcirc\!\!\frown\!\!\bigcirc$-coalgebra

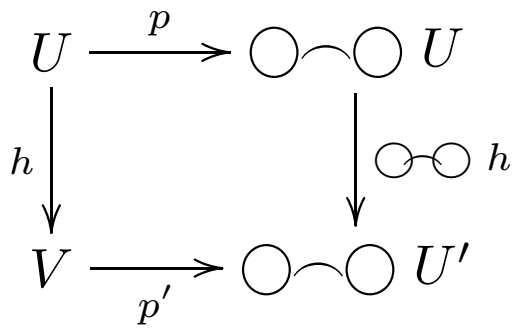
Going Generic

- Mealy machines (on I and O) were introduced as systems observed through

$$\bigcirc\!\frown\!\bigcirc \;=\; (O \times -)^I$$

$\bigcirc\!\frown\!\bigcirc$ as a mapping to decompose the 'observable-universe' U into an 'observation context' $(O \times U)^I$

- The pair $\langle U, p\rangle$ is called a $\bigcirc\!\frown\!\bigcirc$-coalgebra

- 

Going Generic

Interfaces are Functors

A functor is a function over our working universe of typed functions, which which preserves identities and composition

$$\begin{aligned} \mathsf{T}\mathrm{id}_X &= \mathrm{id}_{\mathsf{T}X} \\ \mathsf{T}(f \cdot g) &= \mathsf{T}f \cdot \mathsf{T}g \end{aligned}$$

Going Generic

Interfaces are Functors

A functor is a function over our working universe of typed functions, which which preserves identities and composition

... i.e., the (partial) monoid structure

$$\begin{aligned} \mathsf{T}\mathrm{id}_X &= \mathrm{id}_{\mathsf{T}X} \\ \mathsf{T}(f \cdot g) &= \mathsf{T}f \cdot \mathsf{T}g \end{aligned}$$

Final Coalgebras and Behaviours

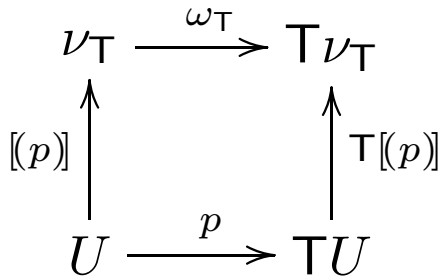
- Any morphism between two T-coalgebras preserves behaviour;

Final Coalgebras and Behaviours

- Any morphism between two T-coalgebras preserves behaviour;
- Whenever the space of behaviours of a class of T-coalgebras can be turned on a T-coalgebra itself

$$\omega_{\mathsf{T}} : \nu_{\mathsf{T}} \longrightarrow \mathsf{T}\nu_{\mathsf{T}}$$

this is the final coalgebra: from any other T-coalgebra p there is a unique morphism $[\![(p)]\!]$ making the following diagram to commute:



Reasoning Coalgebraically

The universal property is equivalently captured by the following law:

$$k = [\![(p)]\!] \quad \Leftrightarrow \quad \omega_{\mathsf{T}} \cdot k = \mathsf{T}\ k \cdot p$$

Reasoning Coalgebraically

The universal property is equivalently captured by the following law:

$$k = [\![(p)]\!] \quad \Leftrightarrow \quad \omega_{\mathsf{T}} \cdot k = \mathsf{T}\ k \cdot p$$

- Existence $\equiv$ definition principle (co-recursion)
- Uniqueness $\equiv$ proof principle (co-induction)

Reasoning Coalgebraically

The universal property is equivalently captured by the following law:

$$k = [\!(p)\!] \quad \Leftrightarrow \quad \omega_{\mathsf{T}} \cdot k = \mathsf{T}\ k \cdot p$$

- Existence $\equiv$ definition principle (co-recursion)
- Uniqueness $\equiv$ proof principle (co-induction)

From which:

$$\begin{array}{rrcl} \text{cancellation} & \omega_{\mathsf{T}} \cdot [\!(p)\!] & = & \mathsf{T}\ [\!(p)\!] \cdot p \\ \text{reflection} & [\!(\omega_{\mathsf{T}})\!] & = & \mathsf{id}_{\nu_{\mathsf{T}}} \\ \text{fusion} & [\!(p)\!] \cdot h & = & [\!(q)\!] \quad \text{if} \quad p \cdot h = \mathsf{T}\ h \cdot q \end{array}$$

Reasoning Coalgebraically

The universal property is equivalently captured by the following law:

$$k = [\!(p)\!] \quad \Leftrightarrow \quad \omega_{\mathsf{T}} \cdot k = \mathsf{T}\ k \cdot p$$

- Existence $\equiv$ definition principle (co-recursion)
- Uniqueness $\equiv$ proof principle (co-induction)

From which:

$$\begin{array}{rrcl} \text{cancellation} & \omega_{\mathsf{T}} \cdot [\!(p)\!] & = & \mathsf{T}\ [\!(p)\!] \cdot p \\ \text{reflection} & [\!(\omega_{\mathsf{T}})\!] & = & \mathsf{id}_{\nu_{\mathsf{T}}} \\ \text{fusion} & [\!(p)\!] \cdot h & = & [\!(q)\!] \quad \text{if} \quad p \cdot h = \mathsf{T}\ h \cdot q \end{array}$$

Reasoning Coalgebraically

The universal property is equivalently captured by the following law:

$$k = [\!(p)\!] \quad \Leftrightarrow \quad \omega_{\mathsf{T}} \cdot k = \mathsf{T}\ k \cdot p$$

- Existence $\equiv$ definition principle (co-recursion)
- Uniqueness $\equiv$ proof principle (co-induction)

From which:

$$\begin{array}{rrcl} \text{cancellation} & \omega_{\mathsf{T}} \cdot [\!(p)\!] & = & \mathsf{T}\ [\!(p)\!] \cdot p \\ \text{reflection} & [\!(\omega_{\mathsf{T}})\!] & = & \mathsf{id}_{\nu_{\mathsf{T}}} \\ \text{fusion} & [\!(p)\!] \cdot h & = & [\!(q)\!] \quad \text{if} \quad p \cdot h = \mathsf{T}\ h \cdot q \end{array}$$

Reasoning Coalgebraically

The universal property is equivalently captured by the following law:

$$k = [\!(p)\!] \quad \Leftrightarrow \quad \omega_{\mathsf{T}} \cdot k = \mathsf{T}\ k \cdot p$$

- Existence $\equiv$ definition principle (co-recursion)
- Uniqueness $\equiv$ proof principle (co-induction)

From which:

$$\begin{array}{rrcl} \text{cancellation} & \omega_{\mathsf{T}} \cdot [\!(p)\!] & = & \mathsf{T}\ [\!(p)\!] \cdot p \\ \text{reflection} & [\!(\omega_{\mathsf{T}})\!] & = & \mathsf{id}_{\nu_{\mathsf{T}}} \\ \text{fusion} & [\!(p)\!] \cdot h & = & [\!(q)\!] \quad \text{if} \quad p \cdot h = \mathsf{T}\ h \cdot q \end{array}$$

Reasoning Coalgebraically

The universal property is equivalently captured by the following law:

$$k = [\!(p)\!] \quad \Leftrightarrow \quad \omega_{\mathsf{T}} \cdot k = \mathsf{T}\ k \cdot p$$

- Existence $\equiv$ definition principle (co-recursion)
- Uniqueness $\equiv$ proof principle (co-induction)

From which:

$$\begin{array}{rrcl} \text{cancellation} & \omega_{\mathsf{T}} \cdot [\!(p)\!] & = & \mathsf{T}\ [\!(p)\!] \cdot p \\ \text{reflection} & [\!(\omega_{\mathsf{T}})\!] & = & \mathsf{id}_{\nu_{\mathsf{T}}} \\ \text{fusion} & [\!(p)\!] \cdot h & = & [\!(q)\!] \quad \text{if} \quad p \cdot h = \mathsf{T}\ h \cdot q \end{array}$$

Example: Lambek's Lemma

The dynamics of the final coalgebra is an isomorphism

proof idea:

- Assume the existence of an inverse α_{T} to $\omega_{\mathsf{T}} : \nu_{\mathsf{T}} \longrightarrow \mathsf{T}\nu_{\mathsf{T}}$.
  Then, $\alpha_{\mathsf{T}} \cdot \omega_{\mathsf{T}} = \mathrm{id}_{\nu_{\mathsf{T}}}$ and $\omega_{\mathsf{T}} \cdot \alpha_{\mathsf{T}} = \mathrm{id}_{\mathsf{T}\nu_{\mathsf{T}}}$

Example: Lambek's Lemma

The dynamics of the final coalgebra is an isomorphism

proof idea:

- Assume the existence of an inverse α_{T} to $\omega_{\mathsf{T}} : \nu_{\mathsf{T}} \longrightarrow \mathsf{T}\nu_{\mathsf{T}}$.
  Then, $\alpha_{\mathsf{T}} \cdot \omega_{\mathsf{T}} = \mathrm{id}_{\nu_{\mathsf{T}}}$ and $\omega_{\mathsf{T}} \cdot \alpha_{\mathsf{T}} = \mathrm{id}_{\mathsf{T}\nu_{\mathsf{T}}}$
- Take one of this requirements and use it to conjecture a definition for α_{T} (or an implementation ...)
  Note the use of the reflection law to introduce an anamorphism in the calculation, instead of eliminating one

Example: Lambek's Lemma

The dynamics of the final coalgebra is an isomorphism

proof idea:

- Assume the existence of an inverse α_{T} to $\omega_{\mathsf{T}} : \nu_{\mathsf{T}} \longrightarrow \mathsf{T}\nu_{\mathsf{T}}$.
  Then, $\alpha_{\mathsf{T}} \cdot \omega_{\mathsf{T}} = \mathrm{id}_{\nu_{\mathsf{T}}}$ and $\omega_{\mathsf{T}} \cdot \alpha_{\mathsf{T}} = \mathrm{id}_{\mathsf{T}\nu_{\mathsf{T}}}$
- Take one of this requirements and use it to conjecture a definition for α_{T} (or an implementation ...)
  Note the use of the reflection law to introduce an anamorphism in the calculation, instead of eliminating one
- Then check the validity of this conjecture by verifying with it the other requirement

Example: the conjecture step

$$\alpha_{\mathsf{T}} \cdot \omega_{\mathsf{T}} = \mathsf{id}_{\nu_{\mathsf{T}}}$$

$=$ { reflection law }

$$\alpha_{\mathsf{T}} \cdot \omega_{\mathsf{T}} = [\!(\omega_{\mathsf{T}})\!]$$

$=$ { anamorphism universal characterisation }

$$\omega_{\mathsf{T}} \cdot \alpha_{\mathsf{T}} \cdot \omega_{\mathsf{T}} = \mathsf{T}(\alpha_{\mathsf{T}} \cdot \omega_{\mathsf{T}}) \cdot \omega_{\mathsf{T}}$$

$=$ { as a functor T preserves composition }

$$\omega_{\mathsf{T}} \cdot \alpha_{\mathsf{T}} \cdot \omega_{\mathsf{T}} = \mathsf{T}\alpha_{\mathsf{T}} \cdot \mathsf{T}\omega_{\mathsf{T}} \cdot \omega_{\mathsf{T}}$$

$=$ { cancel ω_{T} from both sides & ana universal}

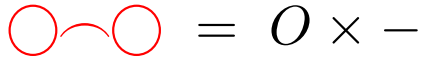
$$\alpha_{\mathsf{T}} = [\!(\mathsf{T}\omega_{\mathsf{T}})\!]$$

Example: the verification step

$$
\begin{array}{ll}
 & \omega_{\mathsf{T}} \cdot \alpha_{\mathsf{T}} \\
= & \quad \{ \text{ replace } \alpha_{\mathsf{T}} \text{ by the derived conjecture } \} \\
 & \omega_{\mathsf{T}} \cdot [\!(\mathsf{T}\omega_{\mathsf{T}})\!] \\
= & \quad \{ \ [\!(\mathsf{T}\omega_{\mathsf{T}})\!] \text{ is a morphism } \} \\
 & \mathsf{T}[\!(\mathsf{T}\omega_{\mathsf{T}})\!] \cdot \mathsf{T}\omega_{\mathsf{T}} \\
= & \quad \{ \text{ as a functor T preserves composition } \} \\
 & \mathsf{T}([\!(\mathsf{T}\omega_{\mathsf{T}})\!] \cdot \omega_{\mathsf{T}}) \\
= & \quad \{ \text{ just proved } \} \\
 & \mathsf{T}\ \mathsf{id}_{\nu_{\mathsf{T}}} \\
= & \quad \{ \text{ as a functor T preserves identities } \} \\
 & \mathsf{id}_{(\mathsf{T}\mathsf{id}_{\nu_{\mathsf{T}}})}
\end{array}
$$

Example: Programming with Streams

Final system for



$$= O \times -$$

Example: Programming with Streams

Final system for

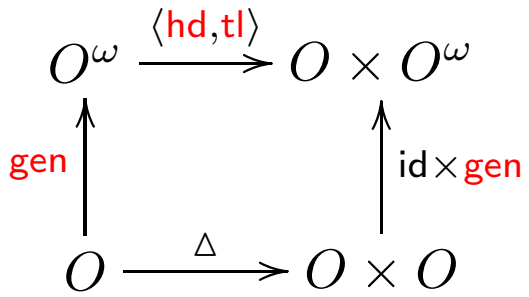
$$= O \times -$$

$$\langle O^\omega, \langle \mathsf{hd}, \mathsf{tl} \rangle : O^\omega \longrightarrow O \times O^\omega \rangle$$

Behaviours $\equiv$ coinductive types

Example: Programming with Streams

Stream Generation



$$\mathsf{gen} = [\![(\Delta)]\!]$$

Δ carries the 'genetic inheritance' of the generating process

From a programming viewpoint it is the eureka! step

Example: Programming with Streams

Coinductive Definition = behaviour specification under all the observers

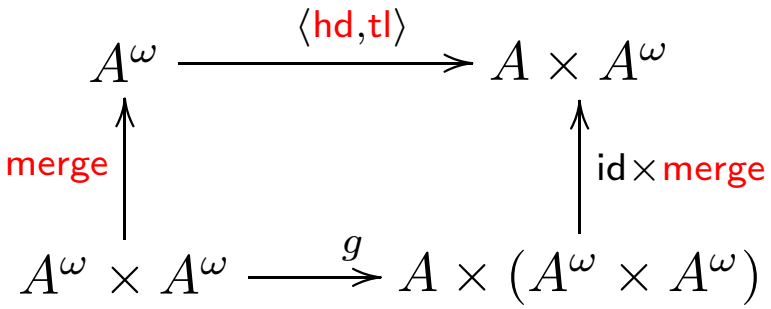
Example: Programming with Streams

Coinductive Definition = behaviour specification under all the observers

$$
\begin{aligned}
& (\mathsf{id} \times \mathsf{gen}) \cdot \triangle = \langle \mathsf{hd}, \mathsf{tl} \rangle \cdot \mathsf{gen} \\
= & \quad \{ \ \triangle \text{ defi nition} \ \} \\
& (\mathsf{id} \times \mathsf{gen}) \cdot \langle \mathsf{id}, \mathsf{id} \rangle = \langle \mathsf{hd}, \mathsf{tl} \rangle \cdot \mathsf{gen} \\
= & \quad \{ \ \times \text{ absorption and fusion} \ \} \\
& \langle \mathsf{id}, \mathsf{gen} \rangle = \langle \mathsf{hd} \cdot \mathsf{gen}, \mathsf{tl} \cdot \mathsf{gen} \rangle \\
= & \quad \{ \ \text{structural equality} \ \} \\
& \mathsf{hd} \cdot \mathsf{gen} = \mathsf{id} \ \wedge \ \mathsf{tl} \cdot \mathsf{gen} = \mathsf{gen} \\
= & \quad \{ \ \text{going pointwise} \ \} \\
& \mathsf{hd} \ (\mathsf{gen} \ a) = a \ \wedge \ \mathsf{tl} \ (\mathsf{gen} \ a) = \mathsf{gen} \ a
\end{aligned}
$$

Example: Programming with Streams

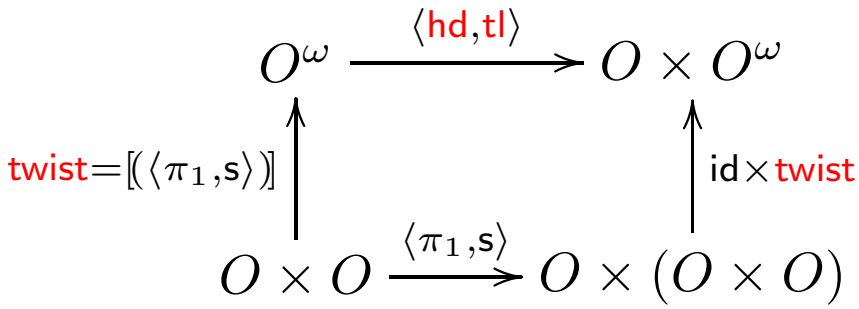
Stream Merge



$$g = \langle \mathsf{hd} \cdot \pi_1, \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id}) \rangle$$

Example: Programming with Streams

Twist



Example: Programming with Streams

$$\boxed{\mathsf{merge}\ (a^\omega, b^\omega) \;=\; (ab)^\omega}$$

i.e.

$$\mathsf{merge} \cdot (\mathsf{gen} \times \mathsf{gen}) \;=\; \mathsf{twist}$$

Example: Streams (Proof)

$$\mathsf{merge} \cdot (\mathsf{gen} \times \mathsf{gen}) \;=\; \mathsf{twist}$$

$=$ { merge definition }

$$[\![(\langle \mathsf{hd} \cdot \pi_1, \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id})\rangle)]\!] \cdot (\mathsf{gen} \times \mathsf{gen}) \;=\; [\![(\langle \pi_1, \mathsf{s}\rangle)]\!]$$

$\Leftarrow$ { anamorphism fusion }

$$\langle \mathsf{hd} \cdot \pi_1, \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id})\rangle \cdot (\mathsf{gen} \times \mathsf{gen}) \;=\; \mathsf{id} \times (\mathsf{gen} \times \mathsf{gen}) \cdot \langle \pi_1, \mathsf{s}\rangle$$

$=$ { $\times$ absorption and reflection }

$$\langle \mathsf{hd} \cdot \mathsf{gen} \cdot \pi_1, \mathsf{s} \cdot ((\mathsf{tl} \cdot \mathsf{gen}) \times \mathsf{gen})\rangle \;=\; \mathsf{id} \times (\mathsf{gen} \times \mathsf{gen}) \cdot \langle \pi_1, \mathsf{s}\rangle$$

$=$ { $\mathsf{tl} \cdot \mathsf{gen} = \mathsf{gen}$ and $\mathsf{hd} \cdot \mathsf{gen} = \mathsf{id}$ }

$$\langle \pi_1, \mathsf{s} \cdot (\mathsf{gen} \times \mathsf{gen})\rangle \;=\; \mathsf{id} \times (\mathsf{gen} \times \mathsf{gen}) \cdot \langle \pi_1, \mathsf{s}\rangle$$

Example: Streams (Proof)

$$\langle \pi_1, \mathsf{s} \cdot (\mathsf{gen} \times \mathsf{gen}) \rangle = \mathsf{id} \times (\mathsf{gen} \times \mathsf{gen}) \cdot \langle \pi_1, \mathsf{s} \rangle$$

$=$ { $\times$ absorption }

$$\langle \pi_1, \mathsf{s} \cdot (\mathsf{gen} \times \mathsf{gen}) \rangle = \langle \pi_1, (\mathsf{gen} \times \mathsf{gen}) \cdot \mathsf{s} \rangle$$

$=$ { s is natural, *i.e.*, $(f \times g) \cdot \mathsf{s} = \mathsf{s} \cdot (g \times f)$ }

$$\langle \pi_1, \mathsf{s} \cdot (\mathsf{gen} \times \mathsf{gen}) \rangle = \langle \pi_1, \mathsf{s} \cdot (\mathsf{gen} \times \mathsf{gen}) \rangle$$

Observational Equivalence

$$u \sim v \iff [\![(p)]\!]\, u = [\![(q)]\!]\, v$$

Observational Equivalence

$$u \sim v \iff [\![(p)]\!]\, u = [\![(q)]\!]\, v$$

- indistinguishability relation on states
- intuition: the final coalgebra is the coalgebra of all behaviours

Observational Equivalence

$$u \sim v \iff [\![p]\!]\, u = [\![q]\!]\, v$$

- indistinguishability relation on states
- intuition: the final coalgebra is the coalgebra of all behaviours

Alternative: two states are equivalent if related by a morphism h (why?)
Then

$$\langle u, v\rangle \in R \;\Rightarrow\; u \sim v$$

where $R = \{\langle x, h\ x\rangle |\ x \in U\}$.

Observational Equivalence

$$u \sim v \iff [\![p]\!]\, u = [\![q]\!]\, v$$

- indistinguishability relation on states
- intuition: the final coalgebra is the coalgebra of all behaviours

Alternative: two states are equivalent if related by a morphism h (why?)
Then

$$\langle u, v\rangle \in R \;\Rightarrow\; u \sim v$$

where $R = \{\langle x, h\ x\rangle |\ x \in U\}$.

Can this idea be generalised?
What properties must a relation R have
to conclude $u \sim v$ by checking if $\langle u, v\rangle$ is in R ?

Example: Streams

Given $p = \langle U, \langle \mathsf{at}_p, \mathsf{m}_p \rangle \rangle$ and $q = \langle V, \langle \mathsf{at}_q, \mathsf{m}_q \rangle \rangle$ and states u and v

when do they generate the same stream?

Example: Streams

Given $p = \langle U, \langle \mathsf{at}_p, \mathsf{m}_p \rangle \rangle$ and $q = \langle V, \langle \mathsf{at}_q, \mathsf{m}_q \rangle \rangle$ and states u and v

when do they generate the same stream?

- Check attributes: $\mathsf{at}_p\ u = \mathsf{at}_q\ v$,
- Assure continuation states $\mathsf{m}_p\ u$ and $\mathsf{m}_q\ v$ support identical verification.

In other words, look for $R \subseteq U \times V$ such that

$$\langle u, v \rangle \in R \ \Rightarrow\ \mathsf{at}_p\ u = \mathsf{at}_q\ v\ \wedge\ \langle \mathsf{m}_p\ u, \mathsf{m}_q\ v \rangle \in R$$

Example: Streams

Given $p = \langle U, \langle \mathsf{at}_p, \mathsf{m}_p \rangle \rangle$ and $q = \langle V, \langle \mathsf{at}_q, \mathsf{m}_q \rangle \rangle$ and states u and v

when do they generate the same stream?

- Check attributes: $\mathsf{at}_p\ u = \mathsf{at}_q\ v$,
- Assure continuation states $\mathsf{m}_p\ u$ and $\mathsf{m}_q\ v$ support identical verification.

In other words, look for $R \subseteq U \times V$ such that

$$\langle u, v \rangle \in R \ \Rightarrow\ \mathsf{at}_p\ u = \mathsf{at}_q\ v\ \wedge\ \langle \mathsf{m}_p\ u, \mathsf{m}_q\ v \rangle \in R$$

, R must be closed under the coalgebra dynamics — a bisimulation

Looking for Genericity

Is there a generic notion of bisimulation (independent of interface T)?

Looking for Genericity

Is there a generic notion of bisimulation (independent of interface T)?

Look for a coalgebra structure in R:

$$\rho = \langle \mathsf{at}_\rho, \mathsf{m}_\rho \rangle : R \longrightarrow O \times R$$

Looking for Genericity

Is there a generic notion of bisimulation (independent of interface T)?

Look for a coalgebra structure in R:

$$\rho = \langle \mathsf{at}_\rho, \mathsf{m}_\rho \rangle : R \longrightarrow O \times R$$

such that

$$\mathsf{at}_\rho = \mathsf{at}_p \cdot \pi_1 = \mathsf{at}_q \cdot \pi_2 \quad \wedge \quad \mathsf{m}_\rho = \mathsf{m}_p \times \mathsf{m}_q$$

Looking for Genericity

A simple calculation yields:

$$\mathsf{at}_\rho = \mathsf{at}_p \cdot \pi_1 = \mathsf{at}_q \cdot \pi_2 \ \wedge \ \mathsf{m}_\rho = \mathsf{m}_p \times \mathsf{m}_q$$

$$\equiv \quad \{ \ \times \text{ reflection} \}$$

$$\mathsf{at}_\rho = \mathsf{at}_p \cdot \pi_1 = \mathsf{at}_q \cdot \pi_2 \ \wedge \ \langle \pi_1, \pi_2 \rangle \cdot \mathsf{m}_\rho = \mathsf{m}_p \times \mathsf{m}_q$$

$$\equiv \quad \{ \ \times \text{ fusion and absorption} \ \}$$

$$\mathsf{at}_\rho = \mathsf{at}_p \cdot \pi_1 = \mathsf{at}_q \cdot \pi_2 \ \wedge \ \langle \pi_1 \cdot \mathsf{m}_\rho, \pi_2 \cdot \mathsf{m}_\rho \rangle = \langle \mathsf{m}_p \cdot \pi_1, \mathsf{m}_q \cdot \pi_2 \rangle$$

$$\equiv \quad \{ \ \text{structural equality} \ \}$$

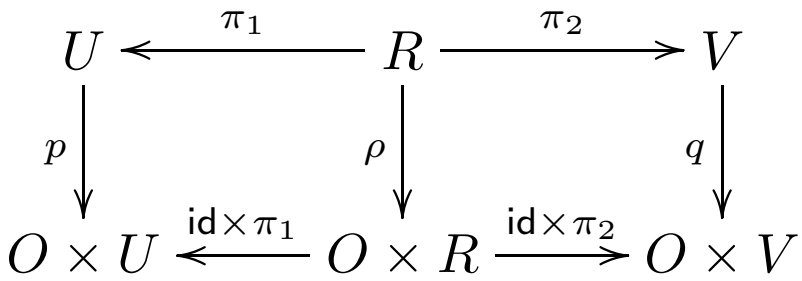
$$\langle \mathsf{at}_\rho, \pi_1 \cdot \mathsf{m}_\rho \rangle = \langle \mathsf{at}_p \cdot \pi_1, \mathsf{m}_p \cdot \pi_1 \rangle \ \wedge \ \langle \mathsf{at}_\rho, \pi_2 \cdot \mathsf{m}_\rho \rangle = \langle \mathsf{at}_q \cdot \pi_2, \mathsf{m}_q \cdot \pi_2 \rangle$$

$$\equiv \quad \{ \ \times \text{ fusion and absorption} \ \}$$

$$(\mathsf{id} \times \pi_1) \cdot \langle \mathsf{at}_\rho, \mathsf{m}_\rho \rangle = \langle \mathsf{at}_p, \mathsf{m}_p \rangle \cdot \pi_1 \ \wedge \ (\mathsf{id} \times \pi_2) \cdot \langle \mathsf{at}_\rho, \mathsf{m}_\rho \rangle = \langle \mathsf{at}_q, \mathsf{m}_q \rangle \cdot \pi_2$$

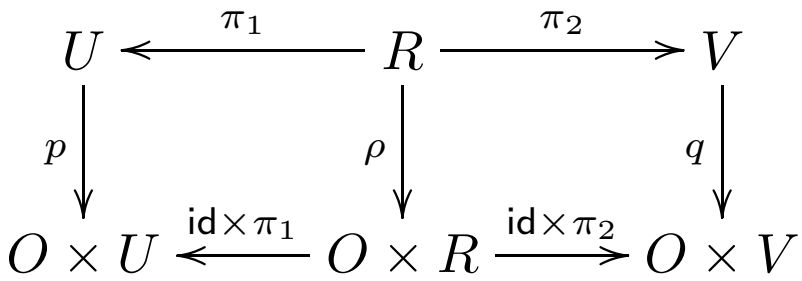
Looking for Genericity

which asserts the commutativity of

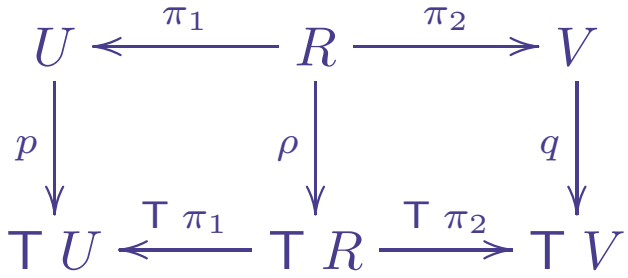


Looking for Genericity

which asserts the commutativity of

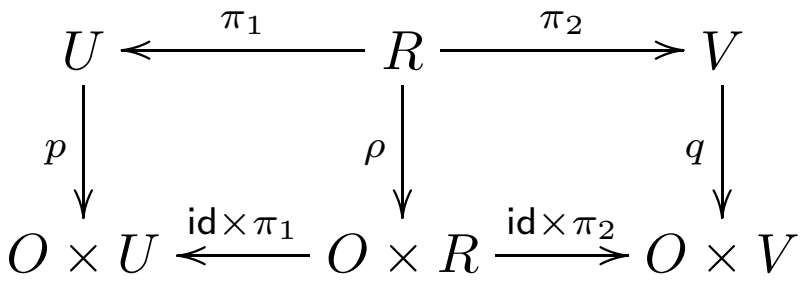


for an arbitrary functor T:

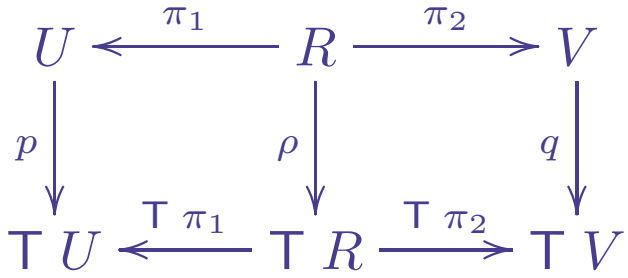


Looking for Genericity

which asserts the commutativity of



for an arbitrary functor T:



$$\mathsf{T}\ \pi_1 \cdot \rho = p \cdot \pi_1 \ \wedge\ \mathsf{T}\ \pi_2 \cdot \rho = q \cdot \pi_2$$

Example: $\mathsf{T} = \mathsf{Id}$

Any relation preserving p and q dynamics is a bisimulation.

Example: $\mathsf{T} = \mathsf{Id}$

Any relation preserving p and q dynamics is a bisimulation.

Because

$$\pi_1 \cdot \rho = p \cdot \pi_1 \ \wedge \ \pi_2 \cdot \rho = q \cdot \pi_2$$

$\equiv$ { structural equality}

$$\langle \pi_1 \cdot \rho, \pi_2 \cdot \rho \rangle \ = \ \langle p \cdot \pi_1, q \cdot \pi_2 \rangle$$

$\equiv$ { $\times$ fusion and absorption}

$$\langle \pi_1, \pi_2 \rangle \cdot \rho \ = \ p \times q \cdot \langle \pi_1, \pi_2 \rangle$$

$\equiv$ { $\times$ reflection }

$$\rho \ = \ p \times q$$

$\equiv$ { going pointwise }

$$\langle u, v \rangle \in R \ \text{ iff } \ u \in U \ \wedge \ v \in V$$

Example: $\mathsf{T} = \mathcal{P}(O \times \mathsf{Id})$

A similar calculation leads to

$$\langle u, v\rangle \in R \;\Rightarrow$$

$$\forall_{\langle o,x\rangle \in p\, u} \cdot \exists_{\langle o,y\rangle \in q\, v} \cdot \langle x, y\rangle \in R$$

$$\wedge$$

$$\forall_{\langle o,y\rangle \in q\, v} \cdot \exists_{\langle o,x\rangle \in p\, u} \cdot \langle x, y\rangle \in R$$

which corresponds to CCS strict bisimulation

T-bisimulation

- Provides a ‘relational’ view of coalgebra morphisms, as the graph of a T-coalgebra morphism is a T-bisimulation.

T-bisimulation

- Provides a ‘relational’ view of coalgebra morphisms, as the graph of a T-coalgebra morphism is a T-bisimulation.
- Has a large theory (*e.g.*, closed for converse and composition)

T-bisimulation

- Provides a ‘relational’ view of coalgebra morphisms, as the graph of a T-coalgebra morphism is a T-bisimulation.
- Has a large theory (*e.g.*, closed for converse and composition)
- Entails a local proof theory for observational equivalence (*cf.*, ‘main’ trend in the literature)

T-bisimulation

- Provides a ‘relational’ view of coalgebra morphisms, as the graph of a T-coalgebra morphism is a T-bisimulation.
- Has a large theory (*e.g.*, closed for converse and composition)
- Entails a local proof theory for observational equivalence (*cf.*, ‘main’ trend in the literature)
- ... to be contrasted with our calculational approach closer to functional programming

T-bisimulation

- Provides a ‘relational’ view of coalgebra morphisms, as the graph of a T-coalgebra morphism is a T-bisimulation.
- Has a large theory (*e.g.*, closed for converse and composition)
- Entails a local proof theory for observational equivalence (*cf.*, ‘main’ trend in the literature)
- ... to be contrasted with our calculational approach closer to functional programming
- Two sides of the same coin: final coalgebras are full abstract with respect to bisimulation

T-bisimulation

- Provides a ‘relational’ view of coalgebra morphisms, as the graph of a T-coalgebra morphism is a T-bisimulation.
- Has a large theory (*e.g.*, closed for converse and composition)
- Entails a local proof theory for observational equivalence (*cf.*, ‘main’ trend in the literature)
- ... to be contrasted with our calculational approach closer to functional programming
- Two sides of the same coin: final coalgebras are full abstract with respect to bisimulation
- Modal reasoning (again parametric on system’s interface T) [Moss99, Jacobs99, Kurz01, ...]

Coinductive Programming

CHARITY [Cockett et al, 92] is based on the term logic of distributive categories with a definitional mechanism for categorical datatypes *i.e.*, for both initial algebras and final coalgebras

Coinductive Programming

- CHARITY [Cockett et al, 92] is based on the term logic of distributive categories with a definitional mechanism for categorical datatypes *i.e.*, for both initial algebras and final coalgebras
- replaces the Cartesian closedness requirement, implicit in the original approaches to categorical types, by the assumption that all datatypes are strong

Coinductive Programming

- CHARITY [Cockett et al, 92] is based on the term logic of distributive categories with a definitional mechanism for categorical datatypes *i.e.*, for both initial algebras and final coalgebras
- replaces the Cartesian closedness requirement, implicit in the original approaches to categorical types, by the assumption that all datatypes are strong
- i.e., the underlying functor T possess a strength [Koc72,CS92]

$$\tau_r^{\mathsf{T}} : \mathsf{T} \times - \Longrightarrow \mathsf{T}(\mathsf{Id} \times -)$$

whose effect is to distribute context along functor T

The CHARITY Language

Primitive Types

final object 1

The CHARITY Language

Primitive Types

final object `1`

product `*` with projections `p0` and `p1`

The CHARITY Language

Primitive Types

final object `1`

product `*` with projections `p0` and `p1`

functions are not values and composition, instead of application, is the fundamental primitive

The CHARITY Language

Primitive Types

final object `1`

product `*` with projections `p0` and `p1`

- functions are not values and composition, instead of application, is the fundamental primitive
- is a total language in the sense that any program has a guarantee of termination: the term representing it always reduces to a head normal form and, therefore, a response is computed

The CHARITY Language

Primitive Types

final object `1`

product `*` with projections `p0` and `p1`

- functions are not values and composition, instead of application, is the fundamental primitive
- is a total language in the sense that any program has a guarantee of termination: the term representing it always reduces to a head normal form and, therefore, a response is computed
- ... either lazily or eagerly depending on the types involved being coinductive or inductive, respectively

The CHARITY Language

Coinductive Types

$$\texttt{data}\ S\texttt{->}T\texttt{(}A\texttt{)} = o_1\texttt{:}S\texttt{->}E_1(A,S)\ \texttt{|}\ \ldots\ \texttt{|}\ o_n\texttt{:}S\texttt{->}E_n(A,S)\texttt{.}$$

The CHARITY Language

Coinductive Types

$$\texttt{data}\ S\texttt{->}T(A)\texttt{ = }o_1\texttt{:}S\texttt{->}E_1(A,S)\ \texttt{|}\ \ldots\ \texttt{|}\ o_n\texttt{:}S\texttt{->}E_n(A,S)\texttt{.}$$

Formally, $T(A)$ is the final coalgebra for a functor T_A determined by a signature of observers:

$$\langle \nu_{\mathsf{T}_A}, \langle o_1, \ldots o_n \rangle : \nu_{\mathsf{T}_A} \longrightarrow \prod_i E_i(A, \nu_{\mathsf{T}_A}) \rangle$$

parametric on A

The CHARITY Language

Examples of Coinductive Types

```
data S -> stream(X) = head: S -> X
                    | tail: S -> S.

data S -> colist(X) = delist: S -> SF(X * S).

data S -> conat = denat: S -> SF(S).

data U -> iTree A = node: U -> A
                  | rg  : U -> U
                  | lf  : U -> U.
```

The CHARITY Language

Examples of Coinductive Types

```
data S -> moor(I,O) = att : S -> O
                    | met : S -> I => S.


data S -> mealy(I,O) = a: S -> I => S * O.
```

The CHARITY Language

Inductive Types

$$\texttt{data}\ T(A) \texttt{->} S = c_1 : E_1(A, S) \texttt{->} S \mid \ldots \mid c_n : E_n(A, S) \texttt{->} S.$$

The CHARITY Language

Inductive Types

$\texttt{data}\ T(A) \texttt{->} S = c_1 \texttt{:} E_1(A,S) \texttt{->} S \ \texttt{|}\ \ldots \ \texttt{|}\ c_n \texttt{:} E_n(A,S) \texttt{->} S\texttt{.}$

$T(A)$ is the initial algebra (again parametric on A) for the functor determined by a signature of constructors:

$$\langle \mu\mathsf{T_A}, [c_1, \ldots, c_n] : \sum_i E_i(A, \mu\mathsf{T_A}) \longrightarrow \mu\mathsf{T_A} \rangle$$

The CHARITY Language

Examples of Inductive Types

```
data SF A -> C = ss: A -> C
               | ff: 1 -> C.

data nat -> C = zero: 1 -> C
              | succ: C -> C.

data list A -> C = nil : 1     -> C
                 | cons: A * C -> C.

data bTree (A, B) -> C = leaf: A           -> C
                       | node: B * (C * C) -> C.
```

The CHARITY Language

Combinators (for coinductive types)

unfold (strong anamorphism)

$$(s, c) \Rightarrow (\,|\; s \Rightarrow o_1 : p_1(s, c) \;|\; \cdots \;|\; o_n : p_n(s, c) \;|\,)$$

where

$$p_i : S \times C \longrightarrow E_i(A, S)$$

where S is the carrier of the source coalgebra and C the context type.

The CHARITY Language

Combinators (for coinductive types)

unfold (strong anamorphism)

$$(s, c) \Rightarrow (\,|\; s \Rightarrow o_1 : p_1(s, c) \;|\; \cdots \;|\; o_n : p_n(s, c) \;|\,)$$

where

$$p_i : S \times C \longrightarrow E_i(A, S)$$

where S is the carrier of the source coalgebra and C the context type.

record (non-recursive unfold)

$$c \Rightarrow (\; o_1 : f_1(c) \;\cdots\; |\; o_n : f_n(c) \;)$$

The CHARITY Language

Examples

```
def gen: int  -> stream(int)
    = w => (| x => head: x
             |      tail: x
             |) w.

def stream_gen{f}: X -> stream(X)
    = x => (| x => head: x
             |      tail: f x
             |) x.
```

The CHARITY Language

Examples

```
def twist: O * O  -> stream(O)
    = w => (| (x,y) => head: x
              |      tail: (y,x)
              |) w.

def merge: stream(O) * stream(O)  -> stream(O)
    = w => (| (x,y) => head: head x
              |      tail: (y, tail x)
              |) w.
```

The CHARITY Language

```
def fibb: int * int  -> stream(int)
 = x => (| (m,n) => head: m
          |           tail: (n, add(m,n))
          |) x.

def genfac: int * int -> stream(int)
 =  (n,m)  => (| (x,y) =>
                |   head: x
                |   tail: (mul(x,y), add(y,1))
                |) (n, m).

def factorial: 1 -> stream(int)
    =  ()  =>  genfac(1,1).
```

The CHARITY Language

Combinators (for inductive types)

fold (strong catamorphism)

$$(s, c) \Rightarrow \{|\ c_1 : s_1 \Rightarrow d_1(s_1, c) |\ \cdots\ |\ c_n : s_n \Rightarrow d_n(s_n, c)\ |\} s$$

where

$$d_i : E_i(A, S) \times C \longrightarrow S$$

where C is the context and S the carrier of the target algebra: 'the *either* of all such d_i'

The CHARITY Language

Combinators (for inductive types)

fold (strong catamorphism)

$$(s, c) \Rightarrow \{|\ c_1 : s_1 \Rightarrow d_1(s_1, c)\ |\ \cdots\ |\ c_n : s_n \Rightarrow d_n(s_n, c)\ |\}\ s$$

where

$$d_i : E_i(A, S) \times C \longrightarrow S$$

where C is the context and S the carrier of the target algebra: 'the *either* of all such d_i'

case (non-recursive fold)

$$(s, c) \Rightarrow \{\ c_1 s_1 \Rightarrow d_1(s_1, c)\ |\ \cdots\ |\ c_n s_n \Rightarrow\ \ d_n(s_n, c)\ \}\ s$$

The CHARITY Language

Examples

```
def len: list A -> nat
    = l       => {| nil : ()      => zero
                  | cons: (_, n) => succ n
                  |} l.
```

```
def reduce{oper: M * M -> M, e: 1 -> M}: list(M) -> M
 =  l  =>  {| nil: ()       => e
             | cons: (a,pr) => oper(a,pr)
             |}  l.
```

The CHARITY Language

Combinators (for both)

map (functorial extension)

$$(t, c) \Rightarrow T\{\ x_1 \Rightarrow h_1(x_1, c) \mid \cdots \mid x_m \Rightarrow h_m(x_m, c)\ \}\ t$$

for $h_i : A_i \times C \longrightarrow A'_i$