A Tutorial on Bialgebras

based on work by Turi, Plotkin, Jacobs, and Kozen

Cornell University

November 16, 2022

Denotational Semantics:
the algebra of program composition

Operational Semantics:
the coalgebra of program execution

Denotational Semantics:
the algebra of program composition

- regular expressions

Operational Semantics:
the coalgebra of program execution

- automata

Denotational Semantics:
the algebra of program composition

- regular expressions
- lambda calculus

Operational Semantics:
the coalgebra of program execution

- automata
- turing machines

A category $\mathcal{C}$ is:

- collection of objects $\mathrm{ob}\,\mathcal{C}$;
- collection of morphisms $\{X \xrightarrow{a} Y\}$ between them
  - that can be composed associatively,
  - including identities $\mathrm{id}_X : X \to X$

A category $\mathcal{C}$ is:

- collection of objects $\operatorname{ob} \mathcal{C}$;
- collection of morphisms $\{X \xrightarrow{a} Y\}$ between them
  - that can be composed associatively,
  - including identities $\mathrm{id}_X : X \to X$

Extreme examples:

- monoid = category with exactly one object;
- pre-order = category with at most one arrow between objects

Everyone's favorite category: **Set**, sets and functions.

Initiality / Finality

Initial Object 0

$$0 \overset{\exists!}{\dashrightarrow} X$$

Initiality / Finality

Initial Object 0

$$0 \overset{\exists!}{\dashrightarrow} X$$

Final Object 1

$$X \overset{\exists!}{\dashrightarrow} 1$$

Initiality / Finality

Initial Object 0	Final Object 1
$0 \overset{\exists!}{\dashrightarrow} X$	$X \overset{\exists!}{\dashrightarrow} 1$

- **in Set**: $\emptyset$ $\{*\}$

Initiality / Finality

	Initial Object 0	Final Object 1
	$0 \overset{\exists!}{\dashrightarrow} X$	$X \overset{\exists!}{\dashrightarrow} 1$
▶ in **Set**:	$\emptyset$	$\{*\}$
▶ in $(S, \leq)$:	$\min S$	$\max S$

Quick Review of Functors & Natural Transformations

Let $\mathcal{C}$ and $\mathcal{D}$ be categories.

Quick Review of Functors & Natural Transformations

Let $\mathcal{C}$ and $\mathcal{D}$ be categories.

- A functor $F : \mathcal{C} \to \mathcal{D}$ maps

$$\begin{array}{lcr} X \in \mathrm{ob}\,\mathcal{C} & \text{to} & F(X) \in \mathrm{ob}\,\mathcal{D} \\ f : X \to Y & \text{to} & Ff : FX \to FY \end{array}$$

Quick Review of Functors & Natural Transformations

Let $\mathcal{C}$ and $\mathcal{D}$ be categories.

- A functor $F : \mathcal{C} \to \mathcal{D}$ maps

$$X \in \operatorname{ob} \mathcal{C} \quad \text{to} \quad F(X) \in \operatorname{ob} \mathcal{D}$$

$$f : X \to Y \quad \text{to} \quad Ff : FX \to FY$$

$$\begin{tikzcd} X \arrow[rr, "g \circ f"] \arrow[dr, "f"'] & & Z \\ & Y \arrow[ur, "g"'] & \end{tikzcd} \quad \overset{F}{\longmapsto} \quad \begin{tikzcd} FX \arrow[rr, "F(g \circ f)"] \arrow[dr, "Ff"'] & & FZ \\ & FY \arrow[ur, "Fg"'] & \end{tikzcd}$$

Quick Review of Functors & Natural Transformations

Let $\mathcal{C}$ and $\mathcal{D}$ be categories.

- A functor $F : \mathcal{C} \to \mathcal{D}$ maps

$$\begin{array}{ccc} X \in \mathrm{ob}\,\mathcal{C} & \text{to} & F(X) \in \mathrm{ob}\,\mathcal{D} \\ f : X \to Y & \text{to} & Ff : FX \to FY \end{array}$$

- a natural transformation $\lambda : F \Rightarrow G$ is a family of maps

$$\{\lambda_X : F(X) \to G(X)\}_{X \in \mathrm{ob}\,\mathcal{C}}$$

that commutes with morphisms of $\mathcal{C}$

Quick Review of Functors & Natural Transformations

Let $\mathcal{C}$ and $\mathcal{D}$ be categories.

- A functor $F : \mathcal{C} \to \mathcal{D}$ maps

$$\begin{array}{ccc} X \in \mathrm{ob}\,\mathcal{C} & \text{to} & F(X) \in \mathrm{ob}\,\mathcal{D} \\ f : X \to Y & \text{to} & Ff : FX \to FY \end{array}$$

- a natural transformation $\lambda : F \Rightarrow G$ is a family of maps

$$\{\lambda_X : F(X) \to G(X)\}_{X \in \mathrm{ob}\,\mathcal{C}}$$

  that commutes with morphisms of $\mathcal{C}$

$$\text{for all } f : X \to Y, \quad \begin{array}{ccc} FX & \xrightarrow{Ff} & FY \\ {\scriptstyle \lambda_X}\downarrow & & \downarrow{\scriptstyle \lambda_Y} \\ GX & \xrightarrow[Gf]{} & GY \end{array}$$

Algebra

Given a functor $F : \mathcal{C} \to \mathcal{C}$,
An F-algebra $(X,\ \alpha)$ consists of

- a carrier object $X \in \operatorname{ob} \mathcal{C}$
- a structure map $\alpha : X \to FX$

Example. Take $F = (-)^2 + 1$.
A monoid $(M, \odot, e)$ is a set M together with maps

$$\odot : M \times M \to M, \quad e : \{*\} \to M \qquad \text{(\& associative, unital)}$$

or altogether, an F-algebra

$$\Big(M,\ \langle \odot, e\rangle : (M \times M \sqcup \{*\}) \to M\Big)$$

Example. a Kleene Algebra is a set S with operations

$$+, \cdot : S \times S \to S, \quad 0, 1 : 1 \to S \quad \text{(satisfying some eqns)}$$

Zipped together, they form a Σ-algebra

$$\Big(S,\ [+, \cdot, {}^*, 0, 1] : S^2 + S^2 + S + 1 + 1 \ \to\ S\Big)$$

for the signature Σ of semiring operations

$$\begin{array}{rcl} \Sigma X & ::= & x_1 + x_2 \mid x_1 \cdot x_2 \mid\ x^* \ \mid 0 \mid 1 \\ & = & X^2 \ + \ X^2 \ + X + 1 + 1 \end{array}$$

Example. a Kleene Algebra is a set S with operations

$$+, \cdot : S \times S \to S, \quad 0, 1 : 1 \to S \quad \text{(satisfying some eqns)}$$

Zipped together, they form a Σ-algebra

$$\Big(S,\ [+, \cdot, {}^*, 0, 1] : S^2 + S^2 + S + 1 + 1 \ \to\ S\Big)$$

for the signature Σ of semiring operations

$$\begin{array}{rl} \Sigma X ::= & x_1 + x_2 \mid x_1 \cdot x_2 \mid\ x^* \mid 0 \mid 1 \\ = & X^2 \ + \ X^2 \ + X + 1 + 1 \end{array}$$

- The initial Σ-algebra is the set of regular expressions.

Quick review of Monads

A monad $T = (T, \eta, \mu)$ is

- a functor $T : \mathcal{C} \to \mathcal{C}$,
- multiplication $\mu : T^2 \Rightarrow T$
  that is associative

$$\begin{tikzcd}
a;((b;c);d) \in & T^3X \arrow[r, "\mu_{TX}"] \arrow[d, "T\mu_X"] & T^2X \arrow[d, "\mu_X"'] & \ni a;(b;c);d \\
a;(b;c;d) \in & T^2X \arrow[r, "\mu_X"] & TX & \ni a;b;c;d
\end{tikzcd}$$

- a unit $\eta : 1 \Rightarrow T$

$$\begin{tikzcd}
(a) \in & TX \arrow[r, "T(\eta_X)"] \arrow[dr, equal] & T^2X \arrow[d, "\mu_X"] & \ni ((a)) \\
& & TX & \ni (a)
\end{tikzcd}$$

Algebras of a Monad

- Like before, $(X,\ \alpha : TX \to X)$.

Algebras of a Monad

- Like before, $(X,\ \alpha : TX \to X)$.
- But also

$$
\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[dr, equal] & TX \arrow[d, "\alpha"] \\
& X
\end{tikzcd}
\qquad
\begin{tikzcd}
TX \arrow[d, "\alpha"] & TTX \arrow[l, "T\alpha"'] \arrow[d, "\mu_X"] \\
X & TX \arrow[l, "\alpha"]
\end{tikzcd}
$$

Algebras of a Monad

- Like before, $(X,\ \alpha : TX \to X)$.
- But also

$$\begin{tikzcd} X \arrow[r, "\eta_X"] \arrow[dr, equal] & TX \arrow[d, "\alpha"] \\ & X \end{tikzcd} \qquad \begin{tikzcd} TX \arrow[d, "\alpha"] & TTX \arrow[l, "T\alpha"'] \arrow[d, "\mu_X"] \\ X & TX \arrow[l, "\alpha"] \end{tikzcd}$$

- For example:

$$\begin{tikzcd} \mathbb{R} \arrow[r, "{[-]}"] \arrow[dr, equal] & \mathsf{List}[\mathbb{R}] \arrow[d, "\mathsf{sum}"] \\ & \mathbb{R} \end{tikzcd} \qquad \begin{tikzcd} \mathsf{List}[\mathbb{R}] \arrow[d, "\mathsf{sum}"] & \mathsf{List}[\mathsf{List}[\mathbb{R}]] \arrow[l, "\mathsf{map}(\mathsf{sum})"'] \arrow[d, "\mathsf{flatten}"] \\ \mathbb{R} & \mathsf{List}[\mathbb{R}] \arrow[l, "\mathsf{sum}"] \end{tikzcd}$$

Algebras of a Monad

- Like before, $(X,\ \alpha : TX \to X)$.
- But also

$$
\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[dr, equal] & TX \arrow[d, "\alpha"] \\
 & X
\end{tikzcd}
\qquad
\begin{tikzcd}
TX \arrow[d, "\alpha"] & TTX \arrow[l, "T\alpha"'] \arrow[d, "\mu_X"] \\
X & TX \arrow[l, "\alpha"]
\end{tikzcd}
$$

- Eilenberg-Moore category $\mathcal{C}^T$ of algebras:

objects:

(X, α)

$$
\begin{tikzcd}
X \arrow[d, "\alpha"'] \\
TX
\end{tikzcd}
$$

Algebras of a Monad

- Like before, $(X,\ \alpha : TX \to X)$.
- But also

$$\begin{tikzcd} X \arrow[r, "\eta_X"] \arrow[dr, equal] & TX \arrow[d, "\alpha"] \\ & X \end{tikzcd} \qquad \begin{tikzcd} TX \arrow[d, "\alpha"] & TTX \arrow[l, "T\alpha"'] \arrow[d, "\mu_X"] \\ X & TX \arrow[l, "\alpha"] \end{tikzcd}$$

- Eilenberg-Moore category $\mathcal{C}^T$ of algebras:

objects: (X, α)

$$\begin{tikzcd} X \arrow[d, "\alpha"'] \\ TX \end{tikzcd}$$

morphisms: (X, α) (Y, β)

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[d, "\alpha"'] & Y \arrow[d, "\beta"] \\ TX \arrow[r, "Tf"] & TY \end{tikzcd}$$

The Term Monad

Can compose programs

The Term Monad

Can compose programs

X

x

The Term Monad

Can compose programs

$$\begin{array}{ccc} X & + & \Sigma X \\ x & & x; y \end{array}$$

The Term Monad

Can compose programs

$$\begin{array}{ccccc} X & + & \Sigma X & + & \Sigma\Sigma X \\ x & & x; y & & x; (y \oplus z) \end{array}$$

The Term Monad

Can compose programs

$$
\begin{array}{ccccccccc}
X & + & \Sigma X & + & \Sigma\Sigma X & + & \cdots & =: & \Sigma^* X \\
x & & x; y & & x; (y \oplus z) & & & &
\end{array}
$$

Σ^* is the free monad generated by the signature Σ.

The Term Monad

Can compose programs

$$\begin{array}{ccccccccc} X & + & \Sigma X & + & \Sigma\Sigma X & + & \cdots & =: & \Sigma^* X \\ x & & x; y & & x; (y \oplus z) & & & & \end{array}$$

Σ^* is the free monad generated by the signature Σ.
More generally, terms $TX := {}^{\Sigma^* X}\!/_{\cong}$ might satisfy equations.

The Term Monad

Can compose programs

$$\begin{array}{ccccccccc} X & + & \Sigma X & + & \Sigma\Sigma X & + & \cdots & =: & \Sigma^* X \\ x & & x;y & & x;(y \oplus z) & & & & \end{array}$$

Σ^* is the free monad generated by the signature Σ.
More generally, terms $TX := {}^{\Sigma^* X}\!/_{\cong}$ might satisfy equations.

There is an initial

T-algebra	Σ-algebra
$T^2 0 \xrightarrow[\cong]{\mu_0} T0$	$\Sigma\Sigma^* 0 \xrightarrow{\cong} \Sigma^* 0$

The Term Monad

Can compose programs

$$\begin{array}{ccccccccc} X & + & \Sigma X & + & \Sigma\Sigma X & + & \cdots & =: & \Sigma^* X \\ x & & x;y & & x;(y \oplus z) & & & & \end{array}$$

Σ^* is the free monad generated by the signature Σ.
More generally, terms $TX := {}^{\Sigma^* X}\!/_{\cong}$ might satisfy equations.

There is an initial

T-algebra	Σ-algebra
$T^2 0 \xrightarrow[\cong]{\mu_0} T0$	$\Sigma\Sigma^* 0 \xrightarrow{\cong} \Sigma^* 0$

Lemma (Lambek 1968)
The structure map of an initial algebra is an isomorphism.

Denotational Semantics

- Interpret programs P as mathematical objects $\langle\!|P|\!\rangle$ (e.g., numbers)

Denotational Semantics

- Interpret programs P as mathematical objects $\langle\!| P |\!\rangle$ (e.g., numbers)
- That is, specify Σ-algebra $(D, \langle\!| \cdot |\!\rangle)$.

Denotational Semantics

- Interpret programs P as mathematical objects $\llparenthesis P \rrparenthesis$ (e.g., numbers)
- That is, specify Σ-algebra $(D, \llparenthesis \cdot \rrparenthesis)$.

$$\begin{tikzcd} \Sigma D \arrow[d, "\llparenthesis \cdot \rrparenthesis"] \\ D \end{tikzcd}$$

Denotational Semantics

- Interpret programs P as mathematical objects $\llparenthesis P \rrparenthesis$ (e.g., numbers)
- That is, specify Σ-algebra $(D, \llparenthesis \cdot \rrparenthesis)$.

$$
\begin{tikzcd}
\Sigma\Sigma^*0 \arrow[d, "\text{initial algebra}"', "\cong"] & \Sigma D \arrow[d, "\llparenthesis \cdot \rrparenthesis"] \\
\Sigma^*0 & D
\end{tikzcd}
$$

Denotational Semantics

- Interpret programs P as mathematical objects $\llparenthesis P \rrparenthesis$ (e.g., numbers)
- That is, specify Σ-algebra $(D, \llparenthesis \cdot \rrparenthesis)$.

$$\begin{tikzcd}
\Sigma\Sigma^*0 \arrow[r, dashed] \arrow[d, "\text{initial algebra}"', "\cong"] & \Sigma D \arrow[d, "\llparenthesis \cdot \rrparenthesis"] \\
\Sigma^*0 \arrow[r, dashed, "\llparenthesis \cdot \rrparenthesis^{\#}"'] & D
\end{tikzcd}$$

Denotational Semantics

- Interpret programs P as mathematical objects $\llparenthesis P \rrparenthesis$ (e.g., numbers)
- That is, specify Σ-algebra $(D, \llparenthesis \cdot \rrparenthesis)$.

$$\begin{tikzcd} \Sigma\Sigma^*0 \arrow[r, dashed] \arrow[d, "\text{initial algebra}"', "\cong"] & \Sigma D \arrow[d, "\llparenthesis\cdot\rrparenthesis"] \\ \Sigma^*0 \arrow[r, dashed, "\llparenthesis\cdot\rrparenthesis^\#"'] & D \end{tikzcd}$$

- Use of initiality corresponds to induction

Coalgebra

Given a functor $G : \mathcal{C} \to \mathcal{C}$,
A G-coalgebra $(X,\ \gamma)$ consists of

- $X \in \mathsf{ob}\,\mathcal{C}$
- $\gamma : X \to GX$

Example. Again, take $G = (-)^2 + 1$.
A G-coalgebra is a (possibly infinite or recursive) set of binary trees closed under subtree

$$\gamma(t) = \begin{cases} * & \text{if } t \text{ is a leaf} \\ (t_1, t_2) & \text{otherwise} \end{cases}$$

Automata as Coalgebras

A DFA $(Q, \mathbb{A}, \delta, \epsilon)$, where

$$\delta : Q \to Q^{\mathbb{A}}, \qquad \epsilon : 2^{Q}$$

is a coalgebra $(Q, \langle \delta, \epsilon \rangle : Q \to G(Q))$ for the signature $G(X) = X^{\mathbb{A}} \times 2$ of finite automata.

Automata as Coalgebras

A DFA $(Q, \mathbb{A}, \delta, \epsilon)$, where

$$\delta : Q \to Q^{\mathbb{A}}, \qquad \epsilon : 2^{Q}$$

is a coalgebra $(Q, \langle \delta, \epsilon \rangle : Q \to G(Q))$ for the signature $G(X) = X^{\mathbb{A}} \times 2$ of finite automata.

The **final** coalgebra $(2^{\mathbb{A}^*}, \langle \epsilon, \delta \rangle)$ is the semantic Brzozowski derivative

$$\epsilon : 2^{\mathbb{A}^*} \to 2 \qquad \delta_a : 2^{\mathbb{A}^*} \to 2^{\mathbb{A}^*}$$

$$\epsilon(B) = \mathbb{1}[\texttt{""} \in B] \qquad \delta_a(B) = \{x \mid ax \in B\}$$

Category of Coalgebras

G-**CoAlg**

objects: G-coalgebras

$$\begin{tikzcd} X \arrow[d, "\gamma"'] \\ GX \end{tikzcd}$$

morphisms $(X,\gamma) \to (Y,\eta)$

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[d, "\gamma"] & Y \arrow[d, "\eta"] \\ GX \arrow[r, "G\gamma"] & GY \end{tikzcd}$$

Category of Coalgebras

G-**CoAlg**

objects: G-coalgebras

$$\begin{tikzcd} X \arrow[d, "\gamma"'] \\ GX \end{tikzcd}$$

morphisms $(X,\gamma) \to (Y,\eta)$

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[d, "\gamma"] & Y \arrow[d, "\eta"] \\ GX \arrow[r, "G\gamma"] & GY \end{tikzcd}$$

Final coalgebra exists if G has finite branching

$$\begin{tikzcd} Z \arrow[d, "\cong"', "\text{final coalgebra}"] \\ GZ \end{tikzcd}$$

Operational Semantics

G-Colgebra $(S, [\![\cdot]\!])$ with program states S,

Operational Semantics

G-Colgebra $(S, [\![\cdot]\!])$ with program states S,

$$
\begin{tikzcd}
S \arrow[d, "{[\![\cdot]\!]}"'] \\
GS
\end{tikzcd}
$$

Operational Semantics

G-Colgebra $(S, [\![\cdot]\!])$ with program states S,

$$\begin{tikzcd} S \arrow[d, "{[\![\cdot]\!]}"'] & Z \arrow[d, "\cong"', "\text{final coalgebra}"] \\ GS & GZ \end{tikzcd}$$

Operational Semantics

G-Colgebra $(S, [\![\,\cdot\,]\!])$ with program states S,

$$
\begin{tikzcd}
S \arrow[r, dashed, "{[\![\,\cdot\,]\!]^{@}}"] \arrow[d, "{[\![\,\cdot\,]\!]}"'] & Z \arrow[d, "\cong"', "\text{final coalgebra}"] \\
GS \arrow[r, dashed] & GZ
\end{tikzcd}
$$

Operational Semantics

G-Colgebra $(S, [\![\cdot]\!])$ with program states S,

$$\begin{tikzcd} S \arrow[r, dashed, "{[\![\cdot]\!]^{@}}"] \arrow[d, "{[\![\cdot]\!]}"'] & Z \arrow[d, "\cong"', "\text{final coalgebra}"] \\ GS \arrow[r, dashed] & GZ \end{tikzcd}$$

Operational Semantics

G-Colgebra $(S, [\![\cdot]\!])$ with program states S,

$$\begin{tikzcd} S \arrow[r, dashed, "{[\![\cdot]\!]^{@}}"] \arrow[d, "{[\![\cdot]\!]}"'] & Z \arrow[d, "\cong"', "\text{final coalgebra}"] \\ GS \arrow[r, dashed] & GZ \end{tikzcd}$$

- Behavior from finality, called coinduction $[\![\cdot]\!]^{@}$

Distributive Laws

- $F, G : \mathcal{C} \to \mathcal{C}$
- distributive law = natural transformation $\lambda : FG \Rightarrow GF$.

Distributive Laws

- $F, G : \mathcal{C} \to \mathcal{C}$
- distributive law = natural transformation $\lambda : FG \Rightarrow GF$.
- when F is a monad, strengthen to "EM law", by requiring

$$
\begin{tikzcd}
F^2G \arrow[r, "F\lambda"] \arrow[d, "\mu G"'] & FGF \arrow[r, "\lambda F"] & GF^2 \arrow[d, "G\mu"] \\
FG \arrow[rr, "\lambda"'] & & GF
\end{tikzcd}
\qquad
\begin{tikzcd}
& G \arrow[dl, "\eta G"'] \arrow[dr, "G\eta"] & \\
FG \arrow[rr, "\lambda"'] & & GF
\end{tikzcd}
$$

Bialgebra

Given a distributive law $\lambda : FG \Rightarrow GF$,
a λ-bialgebra is a triple (X, α, γ) such that

- (X, α) is an F-algebra
- (X, γ) is a G-coalgebra
- λ glues them together, by

$$\begin{tikzcd}
FX \arrow[r, "\alpha"] \arrow[d, "F(\gamma)"'] & X \arrow[r, "\gamma"] & GX \\
FGX \arrow[rr, "\lambda_X"] & & GFX \arrow[u, "G(\alpha)"']
\end{tikzcd}$$

Allows for Liftings

- α becomes a G-coalgebra morphism

$$\begin{tikzcd} FX \arrow[r, "\lambda_X \circ F\gamma"] \arrow[d, "\alpha"] & G(FX) \arrow[d, "G\alpha"] \\ X \arrow[r, "\gamma"'] & G(X) \end{tikzcd}$$

- γ becomes an F-algebra morphism

$$\begin{tikzcd} X \arrow[r, "\alpha"] \arrow[d, "\gamma"] & F(X) \arrow[d, "F\gamma"] \\ GX \arrow[r, "G\alpha \circ \lambda_X"'] & F(GX) \end{tikzcd}$$

Compatible Operational / Denotational Models

$$
\begin{tikzcd}
\Sigma T0 \arrow[r, dashed] \arrow[d, "\text{initial algebra}"', "\cong"] & \Sigma D \arrow[d, "(\!|\cdot|\!)"] \\
T0 \arrow[r, dashed, "(\!|\cdot|\!)^{\#}"] & D
\end{tikzcd}
$$

$$
\begin{tikzcd}
S \arrow[r, dashed, "{[\![\cdot]\!]^{@}}"'] \arrow[d, "{[\![\cdot]\!]}"'] & Z \arrow[d, "\cong"', "\text{final coalgebra}"] \\
GS \arrow[r, dashed] & GZ
\end{tikzcd}
$$

Compatible Operational / Denotational Models

$$
\begin{tikzcd}
\Sigma T0 \arrow[r, dashed] \arrow[d, "\text{initial algebra}"', "\cong"] & \Sigma D \arrow[d, "\llparenthesis \cdot \rrparenthesis"] & \\
T0 \arrow[r, dashed, "\llparenthesis \cdot \rrparenthesis^{\#}"] & D \arrow[d, equal] & \\
 & S \arrow[r, dashed, "\llbracket \cdot \rrbracket^{@}"'] \arrow[d, "\llbracket \cdot \rrbracket"'] & Z \arrow[d, "\cong"', "\text{final coalgebra}"] \\
 & GS \arrow[r, dashed] & GZ
\end{tikzcd}
$$

Compatible Operational / Denotational Models

$$
\begin{tikzcd}
\Sigma T0 \arrow[r, dashed] \arrow[d, "\text{initial algebra}"', "\cong"] & \Sigma D \arrow[r, "\Sigma[\![\cdot]\!]^{@}"] \arrow[d, "(\!|\cdot|\!)"] & \Sigma Z \arrow[dd] \\
T0 \arrow[r, dashed, "(\!|\cdot|\!)^{\#}"] & D \arrow[d, equal] & \\
 & S \arrow[r, dashed, "[\![\cdot]\!]^{@}"'] \arrow[d, "[\![\cdot]\!]"'] & Z \arrow[d, "\cong"', "\text{final coalgebra}"] \\
 & GS \arrow[r, dashed] & GZ
\end{tikzcd}
$$

Compatible Operational / Denotational Models

$$
\begin{tikzcd}
\Sigma T0 \arrow[r, dashed] \arrow[d, "\text{initial algebra}"', "\cong"] & \Sigma D \arrow[r, "\Sigma[\![\cdot]\!]^{@}"] \arrow[d, "\llparenthesis\cdot\rrparenthesis"] & \Sigma Z \arrow[dd] \\
T0 \arrow[r, dashed, "\llparenthesis\cdot\rrparenthesis^{\#}"] \arrow[dd, "\lambda\circ T(!)"'] & D \arrow[d, equal] & \\
 & S \arrow[r, dashed, "[\![\cdot]\!]^{@}"'] \arrow[d, "[\![\cdot]\!]"'] & Z \arrow[d, "\cong"', "\text{final coalgebra}"] \\
GT0 \arrow[r, "G\llparenthesis\cdot\rrparenthesis^{\#}"'] & GS \arrow[r, dashed] & GZ
\end{tikzcd}
$$

Examples

A Clunky Illustration

Seq Composition $\quad FX := X \times X; \quad GX := O^I \quad$ IO Behavior

$$\begin{tikzcd}
\mathsf{Prog} \times \mathsf{Prog} \arrow[r, ";"] \arrow[d, "eval \times eval"'] & \mathsf{Prog} \arrow[r, "eval"] & O^I \arrow[d, equal] \\
O^I \times O^I \arrow[rr, "\lambda = ?"] & & O^I
\end{tikzcd}$$

- If O has a monoid operation $::$, can use

$$\lambda(f, g) := \text{input} \mapsto f(\text{input}) :: g(\text{input})$$

Producer / Consumer

- Suppose $F, G = (-) \times A$

Producer / Consumer

- Suppose $F, G = (-) \times A$
- A bialgebra

$$A \times X \xrightarrow{\text{put}} X \xrightarrow{\text{get}} X \times A$$

Producer / Consumer

- Suppose $F, G = (-) \times A$
- A bialgebra

$$
\begin{tikzcd}
A \times X \arrow[r, "\mathsf{put}"] \arrow[d, "A\times\mathsf{get}"'] & X \arrow[r, "\mathsf{get}"] & X \times A \\
A \times (X \times A) \arrow[rr, "\lambda=?"] & & (A \times X) \times A \arrow[u, "\mathsf{put}\times A"']
\end{tikzcd}
$$

Producer / Consumer

- Suppose $F, G = (-) \times A$
- A bialgebra

$$\begin{tikzcd}
A \times X \arrow[r, "\mathsf{put}"] \arrow[d, "A\times\mathsf{get}"'] & X \arrow[r, "\mathsf{get}"] & X \times A \\
A \times (X \times A) \arrow[rr, "\lambda=?"] & & (A \times X) \times A \arrow[u, "\mathsf{put}\times A"']
\end{tikzcd}$$

  - is a queue if $\lambda = \mathrm{id}$

Producer / Consumer

- Suppose $F, G = (-) \times A$
- A bialgebra

$$\begin{tikzcd} A \times X \arrow[r, "\mathsf{put}"] \arrow[d, "A\times\mathsf{get}"'] & X \arrow[r, "\mathsf{get}"] & X \times A \\ A \times (X \times A) \arrow[rr, "\lambda=?"] & & (A \times X) \times A \arrow[u, "\mathsf{put}\times A"'] \end{tikzcd}$$

  - is a queue if $\lambda = \mathsf{id}$
  - is a stack if $\lambda = \mathsf{swap}$

KA Bialgebras

The bialgebraic relationship between finite automata and regular expressions:

- $\mathbb{A}$:= a finite alphabet,
- $F(X) = \text{RExp}_{\mathbb{A}} X$, regular expresssions over elements of X and letters of the alphabet;
- $G(X) = 2 \times (-)^{\mathbb{A}}$, the signature of finite automata
- Distributive law: (a slight generalization of) the syntactic Brzowski Derivative

KA Bialgebras

KA Bialgebras

The (Syntactic) Brzozowski Derivative:

a coalgebra $\mathrm{RExp}_{\mathbb{A}} \to 2 \times (\mathrm{RExp}_{\mathbb{A}})^{\mathbb{A}}$

KA Bialgebras

The (Syntactic) Brzozowski Derivative:

a coalgebra $\mathsf{RExp}_{\mathbb{A}} \to 2 \times (\mathsf{RExp}_{\mathbb{A}})^{\mathbb{A}}$

$$E : \mathsf{RExp}_{\mathbb{A}} \to 2 \qquad D_a : \mathsf{RExp}_{\mathbb{A}} \to \mathsf{RExp}_{\mathbb{A}}, \text{ for } a \in \mathbb{A}$$

KA Bialgebras

The (Syntactic) Brzozowski Derivative:
a coalgebra $\mathsf{RExp}_{\mathbb{A}} \to 2 \times (\mathsf{RExp}_{\mathbb{A}})^{\mathbb{A}}$

$$E : \mathsf{RExp}_{\mathbb{A}} \to 2 \qquad D_a : \mathsf{RExp}_{\mathbb{A}} \to \mathsf{RExp}_{\mathbb{A}}, \text{ for } a \in \mathbb{A}$$

$$\begin{array}{ll}
E(e_1 + e_2) = E(e_1) + E(e_2) & D_a(e_1 + e_2) = D_a(e_1) + D_a(e_2) \\
E(e_1 e_2) = E(e_1) \cdot E(e_2) & D_a(e_1 e_2) = D_a(e_1) e_2 + E(e_1) D_a(e_2) \\
E(e^*) = 1 & D_a(e^*) = D_a(e) e^* \\
E(1) = 1 & D_a(1) = D_a(0) = 0 \\
E(0) = E(a) = 0, \text{ for } a \in \mathbb{A} & D_a(b) = \mathbb{1}[b = a], \text{ for } a, b \in \mathbb{A}
\end{array}$$

KA Bialgebras

The (Syntactic) Brzozowski Derivative:

a coalgebra $\mathsf{RExp}_{\mathbb{A}} \to 2 \times (\mathsf{RExp}_{\mathbb{A}})^{\mathbb{A}}$

$$E : \mathsf{RExp}_{\mathbb{A}} \to 2 \qquad D_a : \mathsf{RExp}_{\mathbb{A}} \to \mathsf{RExp}_{\mathbb{A}}, \text{ for } a \in \mathbb{A}$$

$$\begin{array}{ll}
E(e_1 + e_2) = E(e_1) + E(e_2) & D_a(e_1 + e_2) = D_a(e_1) + D_a(e_2) \\
E(e_1 e_2) = E(e_1) \cdot E(e_2) & D_a(e_1 e_2) = D_a(e_1)e_2 + E(e_1)D_a(e_2) \\
E(e^*) = 1 & D_a(e^*) = D_a(e)e^* \\
E(1) = 1 & D_a(1) = D_a(0) = 0 \\
E(0) = E(a) = 0, \text{ for } a \in \mathbb{A} & D_a(b) = \mathbb{1}[b = a], \text{ for } a, b \in \mathbb{A}
\end{array}$$

- $L(e) = \{$language represented by $e\}$ is the unique coalgebra morphism $L : \mathsf{RExp}_{\mathbb{A}} \to 2^{\mathbb{A}^*}$
- used in Brzozowski's proof of Kleene's theorem

KA Bialgebras

$$\mathsf{Brz} : \mathsf{RExp}_{\mathbb{A}}(2 \times (-)^{\mathbb{A}}) \to 2 \times (\mathsf{RExp}_{\mathbb{A}}(-))^{\mathbb{A}}$$

usually presented in curried form

$$E : \mathsf{RExp}_{\mathbb{A}}(2 \times (-)^{\mathbb{A}}) \to 2 \qquad D_a : \mathsf{RExp}_{\mathbb{A}}(2 \times (-)^{\mathbb{A}}) \to \mathsf{RExp}_{\mathbb{A}}(-),\ a \in \mathbb{A}$$

$$\begin{array}{ll}
E(e_1 + e_2) = E(e_1) + E(e_2) & D_a(e_1 + e_2) = D_a(e_1) + D_a(e_2) \\
E(e_1 e_2) = E(e_1)E(e_2) & D_a(e_1 e_2) = D_a(e_1)e_2 + E(e_1)D_a(e_2) \\
E(e^*) = 1 & D_a(e^*) = D_a(e)e^* \\
E(0) = E(a) = 0 & D_p(0) = D_a(1) = 0 \\
E(1) = 1 & D_a(b) = [a = b] \\
\color{red}{E(i, f) = i} & \color{red}{D_a(i, f) = f(a)}
\end{array}$$

KA Bialgebras

The bialgebra diagram becomes:

$$\begin{tikzcd}
\mathsf{RExp}_{\mathbb{A}} X \arrow[r, "\alpha"] \arrow[d, "\mathsf{RExp}_{\mathbb{A}}\langle\epsilon,\delta\rangle"'] & X \arrow[r, "\langle\epsilon,\delta\rangle"] & 2 \times X^{\mathbb{A}} \\
\mathsf{RExp}_{\mathbb{A}}(2 \times X^{\mathbb{A}}) \arrow[rr, "\mathsf{Brz}_X"'] & & 2 \times (\mathsf{RExp}_{\mathbb{A}} X)^{\mathbb{A}} \arrow[u, "\mathsf{id}_2 \times (\alpha)^{\mathbb{A}}"']
\end{tikzcd}$$

Intuitively:

- given a way $\langle \epsilon, \delta \rangle$ of taking derivatives on X,
- can replace each x with $(\epsilon(x), \delta(x))$ derivative, so that
- Brz_X combines them with the traditional syntactic derivative of the expression

KA Bialgebras

Two Extremal Brz-bialgebras

- initial bialgebra, $X = \mathsf{Reg}_{\mathbb{A}} = \mathsf{RExp}_{\mathbb{A}}\emptyset$, regular subsets of $\mathbb{A}^*$

$$\begin{tikzcd}
\mathsf{RExp}_{\mathbb{A}}\mathsf{Reg}_{\mathbb{A}} \arrow[r, "\alpha"] \arrow[d, "\mathsf{RExp}_{\mathbb{A}}\langle\epsilon,\delta\rangle"'] & \mathsf{Reg}_{\mathbb{A}} \arrow[r, "\langle\epsilon,\delta\rangle"] & 2 \times (\mathsf{Reg}_{\mathbb{A}})^{\mathbb{A}} \\
\mathsf{RExp}_{\mathbb{A}}(2 \times (\mathsf{Reg}_{\mathbb{A}})^{\mathbb{A}}) \arrow[rr, "\mathsf{Brz}_{\mathsf{Reg}_{\mathbb{A}}}"'] & & 2 \times (\mathsf{RExp}_{\mathbb{A}}\mathsf{Reg}_{\mathbb{A}})^{\mathbb{A}} \arrow[u, "\mathsf{id}_2 \times (\alpha)^{\mathbb{A}}"']
\end{tikzcd}$$

- final bialgebra, $X = \mathbb{A}^*$

$$\begin{tikzcd}
\mathsf{RExp}_{\mathbb{A}}(\mathbb{A}^*) \arrow[r, "\alpha"] \arrow[d, "\mathsf{RExp}_{\mathbb{A}}\langle\epsilon,\delta\rangle"'] & X \arrow[r, "\langle\epsilon,\delta\rangle"] & 2 \times (\mathbb{A}^*)^{\mathbb{A}} \\
\mathsf{RExp}_{\mathbb{A}}(2 \times (\mathbb{A}^*)^{\mathbb{A}}) \arrow[rr, "\mathsf{Brz}_{\mathbb{A}^*}"'] & & 2 \times (\mathsf{RExp}_{\mathbb{A}}\mathbb{A}^*)^{\mathbb{A}} \arrow[u, "\mathsf{id}_2 \times (\alpha)^{\mathbb{A}}"']
\end{tikzcd}$$

KA Bialgebras

Explicitly, the final Brz-bialgebra

$$\mathsf{RExp}_{\mathbb{A}}(\mathbb{A}^*) \xrightarrow{\alpha} X \xrightarrow{\langle \epsilon, \delta \rangle} 2 \times (\mathbb{A}^*)^{\mathbb{A}}$$

is given by

KA Bialgebras

Explicitly, the final Brz-bialgebra

$$\mathsf{RExp}_{\mathbb{A}}(\mathbb{A}^*) \xrightarrow{\alpha} X \xrightarrow{\langle \epsilon, \delta \rangle} 2 \times (\mathbb{A}^*)^{\mathbb{A}}$$

is given by the semantic Brzozowski derivative

$$\epsilon : 2^{\mathbb{A}^*} \to 2 \qquad \delta_a : 2^{\mathbb{A}^*} \to 2^{\mathbb{A}^*}$$

$$\epsilon(S) = \mathbb{1}[\texttt{""} \in S] \qquad \delta_a(S) = \{x \mid ax \in S\}$$

KA Bialgebras

Explicitly, the final Brz-bialgebra

$$\mathsf{RExp}_{\mathbb{A}}(\mathbb{A}^*) \xrightarrow{\alpha} X \xrightarrow{\langle \epsilon, \delta \rangle} 2 \times (\mathbb{A}^*)^{\mathbb{A}}$$

is given by the semantic Brzozowski derivative

$$\begin{aligned} &\epsilon : 2^{\mathbb{A}^*} \to 2 && \delta_a : 2^{\mathbb{A}^*} \to 2^{\mathbb{A}^*} \\ &\epsilon(S) = \mathbb{1}[\texttt{""} \in S] && \delta_a(S) = \{x \mid ax \in S\} \end{aligned}$$

and α is as one would expect:

$$\begin{aligned} &\alpha(e_1 + e_2) = \alpha(e_1) \cup \alpha(e_2) && \alpha(0) = \emptyset \\ &\alpha(e_1 e_2) = \{xy \mid x \in \alpha(e_1),\ y \in \alpha(e_2)\} && \alpha(1) = \{\epsilon\} \\ &\alpha(e^*) = \bigcup_n \alpha(e^n) && \alpha(a) = \{a\} \\ & && \alpha(S) = S \end{aligned}$$

Many Variants

- KAT — combine with boolean tests
- NetKAT — NetKAT expressions, and NetKAT packet forwarding model
- GKAT — guarded expressions and fully deterministic automata
- KAT + B! — extend with mutable variables

Many Variants

- KAT — combine with boolean tests
- NetKAT — NetKAT expressions, and NetKAT packet forwarding model
- GKAT — guarded expressions and fully deterministic automata
- KAT + B! — extend with mutable variables

In each case, there is a different syntax F, different behavior G, but always a distributive law (closely related to the Brzozowski Derivative) relating them bialgebraically.

Some Bialgebra Facts

Distributive laws correspond to Liftings over monads

Lemma (Jacobs)

Kl Laws $\lambda : FT \Rightarrow TF$ EM Laws $\lambda : TG \Rightarrow GT$

Distributive laws correspond to Liftings over monads

Lemma (Jacobs)

Kl Laws $\lambda : FT \Rightarrow TF$

EM Laws $\lambda : TG \Rightarrow GT$

$$\Updownarrow$$

$$\begin{tikzcd} \mathcal{EM}(T) \arrow[r] \arrow[d] & \mathcal{EM}(T) \arrow[d] \\ \mathcal{C} \arrow[r, "G"] & \mathcal{C} \end{tikzcd}$$

Distributive laws correspond to Liftings over monads

Lemma (Jacobs)

Kl Laws $\lambda : FT \Rightarrow TF$

$$\Updownarrow$$

$$\begin{tikzcd}
\mathcal{K}\ell(T) \arrow[r] \arrow[d] & \mathcal{K}\ell(T) \arrow[d] \\
\mathcal{C} \arrow[r, "F"] & \mathcal{C}
\end{tikzcd}$$

EM Laws $\lambda : TG \Rightarrow GT$

$$\Updownarrow$$

$$\begin{tikzcd}
\mathcal{EM}(T) \arrow[r] \arrow[d] & \mathcal{EM}(T) \arrow[d] \\
\mathcal{C} \arrow[r, "G"] & \mathcal{C}
\end{tikzcd}$$

For an EM law $\rho : TG \Rightarrow GT$,

$$\mathbf{CoAlg}(\mathcal{EM}(G)) \cong \mathbf{BiAlg}(TG \overset{\rho}{\Rightarrow} GT) \cong \mathcal{EM}(\mathsf{Coalg}(T))$$

Theorem (Jacobs)

If there is an initial object 0 and a final G-coalgebra,

- Algebraic and coalgebraic semantics coincide

Theorem (Jacobs)

If there is an initial object 0 and a final G-coalgebra,

- Algebraic and coalgebraic semantics coincide

$$
\begin{tikzcd}
\Sigma T0 \arrow[r, dashed] \arrow[d, "\text{initial algebra}"', "\cong"] & \Sigma D \arrow[r, "\Sigma[\![\cdot]\!]^{@}"] \arrow[d, "(\!|\cdot|\!)"] & \Sigma Z \arrow[dd] \\
T0 \arrow[r, dashed, "(\!|\cdot|\!)^{\#}"] \arrow[dd, "\lambda \circ T(!)"'] & D \arrow[d, equal] & \\
& S \arrow[r, dashed, "[\![\cdot]\!]^{@}"'] \arrow[d, "[\![\cdot]\!]"'] & Z \arrow[d, "\cong"', "\text{final coalgebra}"] \\
GT0 \arrow[r, "G(\!|\cdot|\!)^{\#}"'] & GS \arrow[r, dashed] & GZ
\end{tikzcd}
$$

Theorem (Jacobs)

If there is an initial object 0 and a final G-coalgebra,

- Algebraic and coalgebraic semantics coincide

$$\begin{tikzcd}
\Sigma T0 \arrow[rr, dashed] \arrow[d, "\text{initial algebra}"', "\cong"] & & \Sigma Z \arrow[d] \\
T0 \arrow[rrd, dashed, "(\!|\cdot|\!)^{\#}", "[\![\cdot]\!]^{@}"'] \arrow[dd, "\lambda\circ T(!)"'] & & \\
& & Z \arrow[d, "\cong"', "\text{final coalgebra}"] \\
GT0 \arrow[r, "G(\!|\cdot|\!)^{\#}"'] & GS \arrow[r, dashed] & GZ
\end{tikzcd}$$

Theorem (Jacobs)

If there is an initial object 0 and a final G-coalgebra,

- Algebraic and coalgebraic semantics coincide

$$
\begin{tikzcd}
\Sigma T0 \arrow[rr, dashed] \arrow[d, "\text{initial algebra}"', "\cong"] & & \Sigma Z \arrow[dd] \\
T0 \arrow[rrd, dashed, "\langle\!|\cdot|\!\rangle^{\#}", "{[\![\cdot]\!]^{@}}"'] \arrow[dd, "\lambda\circ T(!)"'] & & \\
 & & Z \arrow[d, "\cong"', "\text{final coalgebra}"] \\
GT0 \arrow[r, "G\langle\!|\cdot|\!\rangle^{\#}"'] & GS \arrow[r, dashed] & GZ
\end{tikzcd}
$$

- Bisimilarity / observational equivalence on $T0$ is a congruence

Denotational Semantics:
the algebra of program composition

- regular expressions
- lambda calculus

Operational Semantics:
the coalgebra of program execution

- automata
- turing machines