# Why Adjunctions Matter

**WADT 2022**

Aveiro, 30th June 2022

J.N. Oliveira

INESC TEC & University of Minho

# Thanks for inviting!

**Algebraic Development Techniques**:

- WADT 82 ... – (algebraic) **abstract data type** trend
- WADT 92 – Hermida fibred **adjunctions**
- WADT ... – lots of other interesting topics!

Algebraic techniques in this talk:

- **Adjunctions** as central device for reasoning.
- **Galois connections** as one of their most useful instances.

Perspective:

- mathematics of program construction.

# Thanks for inviting!

**Algebraic Development Techniques**:

- WADT 82 ... – (algebraic) **abstract data type** trend
- WADT 92 – Hermida fibred **adjunctions**
- WADT ... – lots of other interesting topics!

Algebraic techniques in this talk:

- **Adjunctions** as central device for reasoning.
- **Galois connections** as one of their most useful instances.

Perspective:

- mathematics of program construction.

# Why Adjunctions Matter

- For the average programmer, **adjunctions** are (if known) more respected than loved.
- However, they are key to explaining many things we do as programmers.
- I will try to show how practical adjunctions are by revealing their "chemistry" in action.
- Starting from **Galois connections**, their simplest (but quite interesting) instances, with applications.

# Inspiration

*"My experience has been that theories are often more structured and more interesting when they are based on the real problems; somehow they are more exciting than completely abstract theories will ever be."*
Donald Knuth (1973)

*"(...) this was agreed upon and Jim Thatcher proposed the name* **ADJ** *as a (terrible) pun on the title of the book that we had planned to write (...) [recalling] that* **adjointness** *is a very important concept in category theory (...)"*

(Joseph A. Goguen, *Memories of ADJ*, EATCS nr. 36, 1989)

# Things come in dichotomies

In everyday life, things come “in pairs”

- good
- bad

- action
- reaction

- the left
- the right

- **easy**
- **hard**

In a sense, each pair defines itself:

- one of its elements exists...
- ... because the other also exists, and is **opposite** to it.

Circularity? We can deal with it.

# Things come in dichotomies

In everyday life, things come "in pairs"

- good
- bad

- action
- reaction

- the left
- the right

- **easy**
- **hard**

In a sense, each pair defines itself:

- one of its elements exists...
- ... because the other also exists, and is **opposite** to it.

Circularity? We can deal with it.

# Things come in dichotomies

In everyday life, things come "in pairs"

- good
- bad

- action
- reaction

- the left
- the right

- **easy**
- **hard**

In a sense, each pair defines itself:

- one of its elements exists...
- ... because the other also exists, and is **opposite** to it.

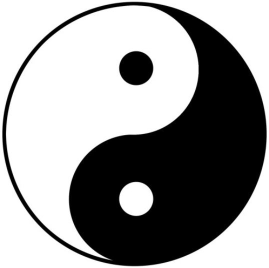
Circularity? We can deal with it.

# Perfect antithesis

The perfect antithesis (opposition, inversion) is the **bijection** or **isomorphism**.

For instance, **multiplying** and **dividing** are inverses of each other in $\boldsymbol{R}$:

$$(x \,/\, y) * y = x$$
$$(x * y) \,/\, y = x$$



---

**Lossless** *transformations:*




$$\begin{cases} f\ (g\ b) = b \\ g\ (f\ a) = a \end{cases}$$

*(Also "energy preserving".)*

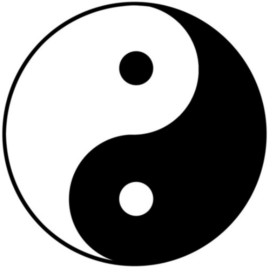
---

# Perfect antithesis

The perfect antithesis (opposition, inversion) is the **bijection** or **isomorphism**.

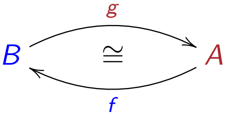
For instance, **multiplying** and **dividing** are inverses of each other in $\mathbb{R}$:

$$(x \,/\, y) * y = x$$
$$(x * y) \,/\, y = x$$



---

**Lossless** *transformations:*

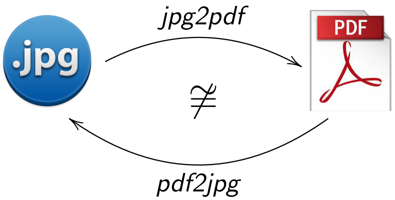



$$\begin{cases} f\ (g\ b) = b \\ g\ (f\ a) = a \end{cases}$$

*(Also "energy preserving".)*

---

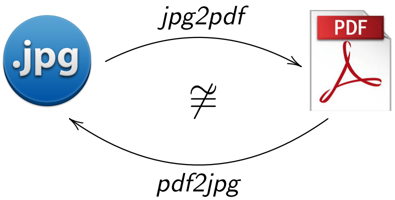
# However, in practice...




$$jpg2pdf \cdot pdf2jpg \neq id$$
$$pdf2jpg \cdot jpg2pdf \neq id$$

(though our eyes can't see the difference in most cases...)

# However, in practice...




$$\mathit{jpg2pdf} \cdot \mathit{pdf2jpg} \neq \mathit{id}$$
$$\mathit{pdf2jpg} \cdot \mathit{jpg2pdf} \neq \mathit{id}$$

(though our eyes can't see the difference in most cases...)

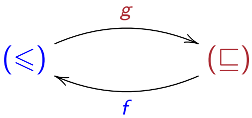
# Lossy inversions

In general, transformations are **lossy**

$$\begin{cases} f\ (g\ x) \leqslant x \\ a \sqsubseteq g\ (f\ a) \end{cases} \tag{1}$$

in the sense that each *"round trip"* loses information.

So we have under and over **approximations** captured by **preorders**:

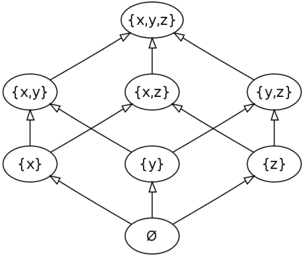



($f$ and $g$ assumed monotonic)

# Handling approximations

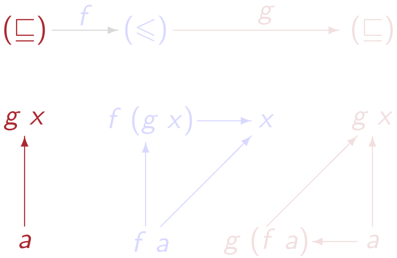
We write $x \xrightarrow{(\leqslant)} y$ (resp. $x \xrightarrow{(\sqsubseteq)} y$ ) to denote $x \leqslant y$ (resp. $x \sqsubseteq y$).

But we drop the orderings, e.g. $x \longrightarrow y$ , wherever these are clear from the context.




**Arrows** *enable us to express our reasoning* **graphically**.

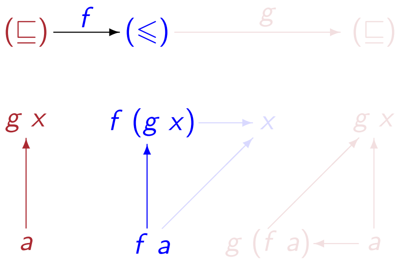
# Handling approximations




$$f\ a \leqslant x \quad \Leftrightarrow \quad a \sqsubseteq g\ x \tag{2}$$

We say $f$ and $g$ are **Galois connected** and write $f \dashv g$ to say so.

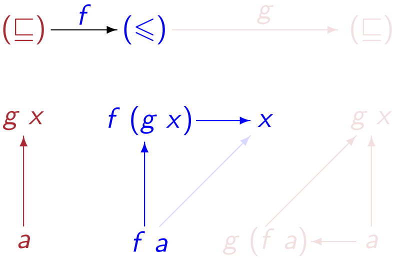
# Handling approximations




$$f\ a \leqslant x \quad \Leftrightarrow \quad a \sqsubseteq g\ x \tag{2}$$

We say $f$ and $g$ are **Galois connected** and write $f \dashv g$ to say so.

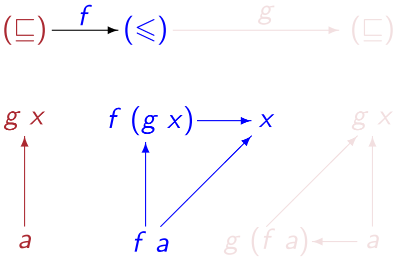
# Handling approximations




$$f\ a \leqslant x \quad \Leftrightarrow \quad a \sqsubseteq g\ x \qquad (2)$$

We say $f$ and $g$ are **Galois connected** and write $f \dashv g$ to say so.

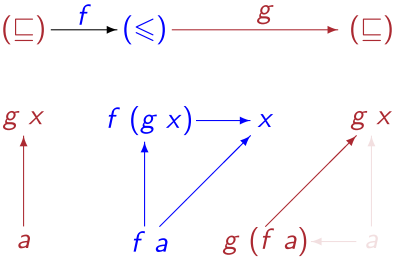
# Handling approximations




$$f\ a \leqslant x \quad \Leftrightarrow \quad a \sqsubseteq g\ x \tag{2}$$

We say $f$ and $g$ are **Galois connected** and write $f \dashv g$ to say so.

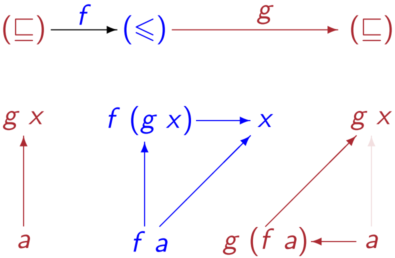
# Handling approximations




$$f\ a \leqslant x \quad \Leftrightarrow \quad a \sqsubseteq g\ x \tag{2}$$

We say $f$ and $g$ are **Galois connected** and write $f \dashv g$ to say so.

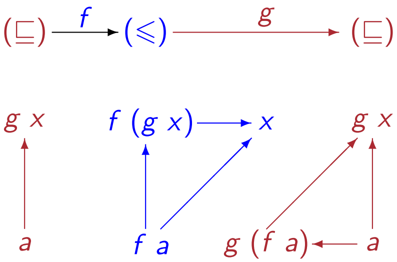
# Handling approximations




$$f\ a \leqslant x \quad \Leftrightarrow \quad a \sqsubseteq g\ x \tag{2}$$

We say $f$ and $g$ are **Galois connected** and write $f \dashv g$ to say so.

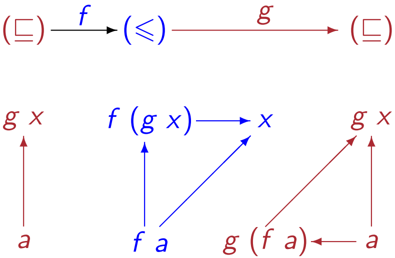
# Handling approximations




$$f\ a \leqslant x \quad \Leftrightarrow \quad a \sqsubseteq g\ x \tag{2}$$

We say $f$ and $g$ are **Galois connected** and write $f \dashv g$ to say so.

# Handling approximations



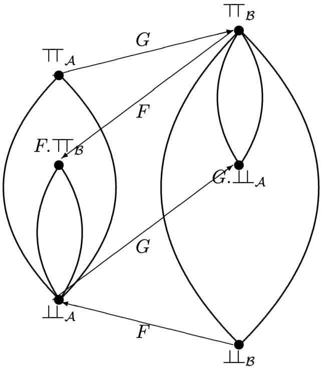

$$f\ a \leqslant x \quad \Leftrightarrow \quad a \sqsubseteq g\ x \tag{2}$$

We say $f$ and $g$ are **Galois connected** and write $f \dashv g$ to say so.

# $f \dashv g$

$$f\ a \leqslant x \quad \Leftrightarrow \quad a \sqsubseteq g\ x$$

- $f$ — **lower** (aka **left**) adjoint
- $g$ — **upper** (aka **right**) adjoint




(Courtesy of R. Backhouse)

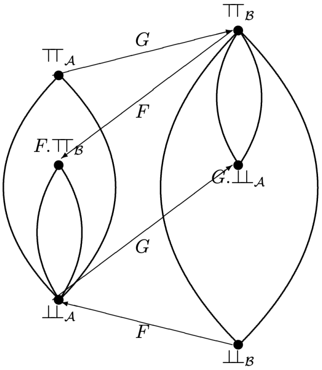
In fact, note the *superlatives* in

- $f\ a$ — **lowest** $x$ such that $a \sqsubseteq g\ x$
- $g\ x$ — **greatest** $a$ such that $f\ a \leqslant x$

# $f \dashv g$

$$f\ a \leqslant x \quad \Leftrightarrow \quad a \sqsubseteq g\ x$$

- $f$ — **lower** (aka **left**) adjoint
- $g$ — **upper** (aka **right**) adjoint




(Courtesy of R. Backhouse)

In fact, note the *superlatives* in

- $f\ a$ — **lowest** $x$ such that $a \sqsubseteq g\ x$
- $g\ x$ — **greatest** $a$ such that $f\ a \leqslant x$

# Handling approximations

Did you say *"superlatives"*?

We have plenty of these in **software requirements**:

> … *the* **largest** *prefix of* $x$ *with at most* $n$ *elements*

($take\ n\ x$, Haskell terminology)

> … *the* **largest** *number that multiplied by* $y$ *is at most* $x$

(integer division $x \div y$).

# Handling approximations

Did you say *"superlatives"*?

We have plenty of these in **software requirements**:

---

... *the* **largest** *prefix of* $x$ *with at most* $n$ *elements*

---

($take\ n\ x$, Haskell terminology)

---

... *the* **largest** *number that multiplied by* $y$ *is at most* $x$

---

(integer division $x \div y$).

# On numeric division

In the reals ($\mathbb{R}$):

$$a \times y = x \quad \Leftrightarrow \quad a = x \, / \, y$$

— an **isomorphism**.

In the natural numbers ($\mathbb{N}_0$):

$$a \times y \leqslant x \quad \Leftrightarrow \quad a \leqslant x \div y$$

— a Galois **connection**.

| $x$ | $y$ |
|---|---|
| ... | $x \div y$ |

$x \div y$
**largest** $a$
such that
$a \times y \leqslant x$.

# On numeric division

In the reals ($\mathbb{R}$):

$$a \times y = x \quad \Leftrightarrow \quad a = x \,/\, y$$

— an **isomorphism**.

In the natural numbers ($\mathbb{N}_0$):

$$a \times y \leqslant x \quad \Leftrightarrow \quad a \leqslant x \div y$$

— a Galois **connection**.

$$\begin{array}{c|c} x & y \\ \cline{2-2} \ldots & x \div y \end{array}$$

$x \div y$
**largest** $a$
such that
$a \times y \leqslant x$.

# The easy and the hard

Whole division **specification**:

$$a \times y \leqslant x \Leftrightarrow a \leqslant x \div y$$

that is:

$$a \underbrace{\times y}_{f} \leqslant x \Leftrightarrow a \leqslant x \underbrace{\div y}_{g}$$

that is:

$$(\times y) \vdash (\div y)$$

**Hard** $(\div y)$ *explained by* **easy** $(\times y)$.

# The easy and the hard

Another example:

*take n xs should yield the* **longest** *possible* **prefix** *of xs not exceeding n in* **length**.

**Specification**:

$$\underbrace{\mathit{length}\ \mathit{ys} \leqslant n \wedge \mathit{ys} \sqsubseteq \mathit{xs}}_{\mathit{easy}} \quad \Leftrightarrow \quad \underbrace{\mathit{ys} \sqsubseteq \mathit{take}\ n\ \mathit{xs}}_{\mathit{hard}} \tag{3}$$

— another **GC**.

# The easy and the hard

Many examples, e.g.

---

*The function takeWhile p xs should yield the* **longest prefix** *of xs whose elements all satisfy predicate p.*

---

and

---

*The function filter p xs should yield the* **longest sublist** *of xs such that all x in such a sublist satisfy predicate p.*

---

**NB:** assuming the sublist ordering $ys \preceq xs$ such that e.g. `"ab"` $\preceq$ `"acb"` holds but `"ab"` $\preceq$ `"bca"` **does not** hold.

# Programming from specifications

Can the well-known **implementation**

$$
\begin{array}{l}
x \div y = \\
\quad \textbf{if } x \geqslant y \\
\quad \textbf{then } 1 + (x - y) \div y \\
\quad \textbf{else } 0
\end{array}
$$

be calculated from the **specification**

$z \times y \leqslant x \Leftrightarrow z \leqslant x \div y$ ?

Ups! Not quite right - subtratction in $\mathbb{N}_0$ is not invertible!

No worry — another **GC** comes to the rescue:

$$a \ominus b \leqslant x \Leftrightarrow a \leqslant x + b$$

# Programming from specifications

Can the well-known **implementation**

$$x \div y = \textbf{if } x \geqslant y \textbf{ then } 1 + (x - y) \div y \textbf{ else } 0$$

be calculated from the **specification**

$$z \times y \leqslant x \Leftrightarrow z \leqslant x \div y \text{ ?}$$

Ups! Not quite right - subtratction in $\mathbb{N}_0$ is not invertible!

No worry — another **GC** comes to the rescue:

$$a \ominus b \leqslant x \Leftrightarrow a \leqslant x + b$$

# Indirect equality

Now another brick in the wall (**partial orders** only):

$$a = b \quad \Leftrightarrow \quad \langle \forall\ z\ ::\ z \leqslant a \Leftrightarrow z \leqslant b \rangle \tag{4}$$

This principle of **indirect equality** blends nicely with **GC**s:

$$z \leqslant g\ a$$
$$\Leftrightarrow \qquad \{\ \ldots\ \}$$
$$\ldots \text{(go to the \textbf{easy} side, do things there and come back)}$$
$$\Leftrightarrow \qquad \{\ \ldots\ \}$$
$$z \leqslant \ldots g \ldots a' \ldots$$
$$:: \qquad \{\ \text{indirect equality}\ \}$$
$$g\ a = \ldots g \ldots a' \ldots$$

# Indirect equality

Now another brick in the wall (**partial orders** only):

$$a = b \quad \Leftrightarrow \quad \langle \forall\ z\ \ ::\ \ z \leqslant a \Leftrightarrow z \leqslant b \rangle \tag{4}$$

This principle of **indirect equality** blends nicely with **GC**s:

$$
\begin{array}{ll}
 & z \leqslant g\ a \\
\Leftrightarrow & \qquad \{\ \dots\ \} \\
 & \dots \text{(go to the } \textbf{easy} \text{ side, do things there and come back)} \\
\Leftrightarrow & \qquad \{\ \dots\ \} \\
 & z \leqslant \dots g \dots a' \dots \\
:: & \qquad \{\ \text{indirect equality}\ \} \\
 & g\ a = \dots g \dots a' \dots
\end{array}
$$

# Example — $x \div y$

Case $x \geqslant y$:

$$
\begin{array}{ll}
& z \leqslant x \div y \\
\Leftrightarrow & \quad \{ \ (\times y) \dashv (\div y) \text{ and } (x \ominus y) + y = x \text{ for } x \geqslant y \ \} \\
& z \times y \leqslant (x \ominus y) + y \\
\Leftrightarrow & \quad \{ \ (\ominus y) \dashv (+y) \ \} \\
& (z \times y) \ominus y \leqslant x \ominus y \\
\Leftrightarrow & \quad \{ \ \text{factoring } y \text{ works also for } \ominus \ \} \\
& (z \ominus 1) \times y \leqslant x \ominus y \\
\Leftrightarrow & \quad \{ \ \text{chain the two } \mathbf{GCs} \ \} \\
& z \leqslant 1 + (x \ominus y) \div y \\
:: & \quad \{ \text{ recursive branch calculated thanks to indirect equality } \} \\
& x \div y = 1 + (x \ominus y) \div y
\end{array}
$$

# Example — *take*

Specification **GC**:

$$\mathit{length}\ \mathit{ys} \leqslant n \wedge \mathit{ys} \sqsubseteq \mathit{xs} \quad \Leftrightarrow \quad \mathit{ys} \sqsubseteq \mathit{take}\ n\ \mathit{xs} \tag{5}$$

Standard implementation (Haskell):

$$\begin{aligned} &\mathit{take}\ 0\ \_ \ = [\,] \\ &\mathit{take}\ \_\ [\,] = [\,] \\ &\mathit{take}\ (n+1)\ (h : \mathit{xs}) = h : \mathit{take}\ n\ \mathit{xs} \end{aligned}$$

The same question again: how to derive the **implementation** of *take* from the **specification**?

# Example — *take*

Before that:

---

*We can derive properties of take* **without** *knowing its implementation.*

---

Example:

---

*What happens if we chain two takes in a row?*

---

We **calculate**

$$(take\ m) \cdot (take\ n)$$

in the next slide.

# Example — *take*

$$
\begin{array}{ll}
 & ys \sqsubseteq take\ m\ (take\ n\ xs) \\
\Leftrightarrow & \quad \{\ \text{GC (5)}\ \} \\
 & length\ ys \leqslant m \wedge ys \sqsubseteq take\ n\ xs \\
\Leftrightarrow & \quad \{\ \text{again GC (5)}\ \} \\
 & length\ ys \leqslant m \wedge length\ ys \leqslant n \wedge ys \sqsubseteq xs \\
\Leftrightarrow & \quad \{\ \text{min GC: } a \leqslant x \wedge a \leqslant y \Leftrightarrow a \leqslant x \ \text{`}min\text{`}\ y\ \} \\
 & length\ ys \leqslant (m\ \text{`}min\text{`}\ n) \wedge ys \sqsubseteq xs \\
\Leftrightarrow & \quad \{\ \text{again GC (5)}\ \} \\
 & ys \leqslant take\ (m\ \text{`}min\text{`}\ n)\ xs \\
:: & \quad \{\ \text{indirect equality}\ \} \\
 & take\ m\ (take\ n\ xs)) = take\ (m\ \text{`}min\text{`}\ n)\ xs
\end{array}
$$

No **induction**

(No implementation yet!)

# Example — *take*

$$
\begin{array}{ll}
 & ys \sqsubseteq take\ m\ (take\ n\ xs) \\
\Leftrightarrow & \quad \{\ \text{GC (5)}\ \} \\
 & length\ ys \leqslant m \wedge ys \sqsubseteq take\ n\ xs \\
\Leftrightarrow & \quad \{\ \text{again GC (5)}\ \} \\
 & length\ ys \leqslant m \wedge length\ ys \leqslant n \wedge ys \sqsubseteq xs \\
\Leftrightarrow & \quad \{\ \text{min GC: } a \leqslant x \wedge a \leqslant y \Leftrightarrow a \leqslant x\ `min`\ y\ \} \\
 & length\ ys \leqslant (m\ `min`\ n) \wedge ys \sqsubseteq xs \\
\Leftrightarrow & \quad \{\ \text{again GC (5)}\ \} \\
 & ys \leqslant take\ (m\ `min`\ n)\ xs \\
:: & \quad \{\ \text{indirect equality}\ \} \\
 & take\ m\ (take\ n\ xs)) = take\ (m\ `min`\ n)\ xs
\end{array}
$$

No **induction** 😊
(No implementation yet!)

# Example — *take*

Now the **implementation** (3 cases):

$take\ 0\ \_ = [\,]$

$ys \sqsubseteq take\ 0\ \_$

$\Leftrightarrow$ { GC }

$length\ ys \leqslant 0 \wedge ys \sqsubseteq \_$

$\Leftrightarrow$ { $length\ [\,] = 0$ }

$ys = [\,]$

$\Leftrightarrow$ { antisymmetry of $(\sqsubseteq)$ }

$ys \sqsubseteq [\,]$

$::$ { indirect equality }

$take\ 0\ \_ = [\,]$

$take\ \_\ [\,] = [\,]$

$ys \sqsubseteq take\ \_\ [\,]$

$\Leftrightarrow$ { GC }

$length\ ys \leqslant \_ \wedge ys \sqsubseteq [\,]$

$\Leftrightarrow$ { $length\ [\,] \leqslant \_$ }

$ys \sqsubseteq [\,]$

$::$ { indirect equality }

$take\ \_\ [\,] = [\,]$

# Example — *take*

Now the **implementation** (3 cases):

$$take\ 0\ \_ = [\,]$$

$$
\begin{array}{ll}
 & ys \sqsubseteq take\ 0\ \_ \\
\Leftrightarrow & \quad \{\ \text{GC}\ \} \\
 & length\ ys \leqslant 0 \wedge ys \sqsubseteq \_ \\
\Leftrightarrow & \quad \{\ length\ [\,] = 0\ \} \\
 & ys = [\,] \\
\Leftrightarrow & \quad \{\ \text{antisymmetry of } (\sqsubseteq)\ \} \\
 & ys \sqsubseteq [\,] \\
:: & \quad \{\ \text{indirect equality}\ \} \\
 & take\ 0\ \_ = [\,]
\end{array}
$$

$$take\ \_\ [\,] = [\,]$$

$$
\begin{array}{ll}
 & ys \sqsubseteq take\ \_\ [\,] \\
\Leftrightarrow & \quad \{\ \text{GC}\ \} \\
 & length\ ys \leqslant \_ \wedge ys \sqsubseteq [\,] \\
\Leftrightarrow & \quad \{\ length\ [\,] \leqslant \_\ \} \\
 & ys \sqsubseteq [\,] \\
:: & \quad \{\ \text{indirect equality}\ \} \\
 & take\ \_\ [\,] = [\,]
\end{array}
$$

# Example — *take*

Finally, the remaining case:

$$take\ (n+1)\ (h:xs) = h : take\ n\ xs$$

We will need the following fact about list-prefixing:

$$s \sqsubseteq (h:t) \Leftrightarrow s = [\,] \vee \langle \exists\, s' \ :\ s = (h:s') :\ s' \sqsubseteq t \rangle \qquad (6)$$

(More about this later.)

$$ys \preceq take\ (n+1)\ (h : xs)$$

$\Leftrightarrow$ { GC (3) ; prefix (6) }

$$length\ ys \leqslant n+1 \wedge (ys = [\,] \vee \langle \exists\ ys' \ :\ ys = (h : ys') :\ ys' \preceq xs \rangle)$$

$\Leftrightarrow$ { distribution ; $length\ [\,] \leqslant n+1$ }

$$ys = [\,] \vee \langle \exists\ ys' \ :\ ys = (h : ys') :\ length\ ys \leqslant n+1 \wedge ys' \preceq xs \rangle$$

$\Leftrightarrow$ { $length\ (h : t) = 1 + length\ t$ }

$$ys = [\,] \vee \langle \exists\ ys' \ :\ ys = (h : ys') :\ length\ ys' \leqslant n \wedge ys' \preceq xs \rangle$$

$\Leftrightarrow$ { GC (3) }

$$ys = [\,] \vee \langle \exists\ ys' \ :\ ys = (h : ys') :\ ys' \preceq take\ n\ xs \rangle$$

$\Leftrightarrow$ { fact (6) }

$$ys \preceq h : take\ n\ xs$$

$::$ { indirect equality over list prefixing ($\sqsubseteq$) }

$$take\ (n+1)\ (h : xs) = h : take\ n\ xs$$

# Nice but...

- Where did we get assumption (6) from?
- How do we *calculate* from **GC**s instead of *proving* from **GC**s?

# Galois connections + indirect equality

- S.-C. Mu and J.N. Oliveira. Programming from Galois connections. *JLAP*, 81(6):680–704, 2012.

- P.F. Silva, J.N. Oliveira. 'Galculator': functional prototype of a Galois connection based proof assistant. PPDP '08, 44–55, 2008.

**Galois connections**

# From GCs to adjunctions

Recall $a \xrightarrow{(\leqslant)} b$ meaning

$$(a, b) \in (\leqslant)$$

that is

$$(\leqslant)\ (a, b) = True$$

that is

$$(\leqslant)\ (a, b) = \{(a, b)\}$$

— singleton set made of one of the pairs of relation $(\leqslant)$.

# From GCs to adjunctions

Now compare

$$(\leqslant)\ (a,b) = \{(a,b)\}$$

with something like (broadening scope):

$$\mathfrak{C}\ (a,b) = \{\ \textit{'things that relate } a \textit{ to } b \textit{ in context } \mathfrak{C}\textit{'}\ \}$$

If such "things" have a **name**, e.g. $m$, we can write $m : a \to b$ to indicate their **type**.

*We land into a* **category** — $\mathfrak{C}$ — *where* $a$ *and* $b$ *are* **objects** *and* $m$ *is a* **morphism**.

# From GCs to adjunctions

Now compare

$$(\leqslant)\ (a, b) = \{(a, b)\}$$

with something like (broadening scope):

$$\mathfrak{C}\ (a, b) = \{\ \textit{'things that relate } a \textit{ to } b \textit{ in context } \mathfrak{C}\textit{'}\ \}$$

If such "things" have a **name**, e.g. $m$, we can write $m : a \to b$ to indicate their **type**.

---

*We land into a* **category** — $\mathfrak{C}$ — *where* $a$ *and* $b$ *are* **objects** *and* $m$ *is a* **morphism**.

---

# Categories

Extremely versatile concept, e.g.

$\mathfrak{C}\,(a, b) =$
$\{$ *'matrices with $a$-many columns and $b$-many rows'* $\}$

or

$\mathfrak{C}\,(a, b) = \{$ *'Haskell functions from type $a$ to type $b$'* $\}$

or

$\mathfrak{C}\,(a, b) = \{$ *'binary relations in $a \times b$'* $\}$

# From preorders to categories

“Dramatic” increase in expressiveness:

| **Preorder** | **Category** |
|---|---|
| Object pair | Morphism |
| Reflexivity | Identity |
| Transitivity | Composition |
| Monotonic function | Functor |
| Equivalence | Isomorphism |
| Pointwise ordering | Natural transformation |
| Closure | Monad |
| Galois connection | Adjunction |
| Indirect equality | Yoneda lemma |

The same **game**, but in the **champions league**

# From preorders to categories

“Dramatic” increase in expressiveness:

| **Preorder** | **Category** |
|---|---|
| Object pair | Morphism |
| Reflexivity | Identity |
| Transitivity | Composition |
| Monotonic function | Functor |
| Equivalence | Isomorphism |
| Pointwise ordering | Natural transformation |
| Closure | Monad |
| Galois connection | Adjunction |
| Indirect equality | Yoneda lemma |

The same **game**, but in the **champions league**

# ("Lossy") natural transformations

Recall our starting point,

$$\begin{cases} f\ (g\ x) \leqslant x \\ a \sqsubseteq g\ (f\ a) \end{cases}$$

which meanwhile we wrote thus:

$$\begin{cases} f\ (g\ x) \longrightarrow x \\ a \longleftarrow g\ (f\ a) \end{cases}$$

Champions league version:

$$\begin{cases} \mathbb{F}\ (\mathbb{G}\ X) \xrightarrow{\epsilon} X \\ A \xleftarrow{\eta} \mathbb{G}\ (\mathbb{F}\ A) \end{cases} \quad (7)$$

where $\mathbb{F}$ and $\mathbb{G}$ are functors.

(More about $\epsilon$ and $\eta$ later.)

# ("Lossy") natural transformations

Recall our starting point,

$$\begin{cases} f\ (g\ x) \leqslant x \\ a \sqsubseteq g\ (f\ a) \end{cases}$$

which meanwhile we wrote thus:

$$\begin{cases} f\ (g\ x) \longrightarrow x \\ a \longleftarrow g\ (f\ a) \end{cases}$$

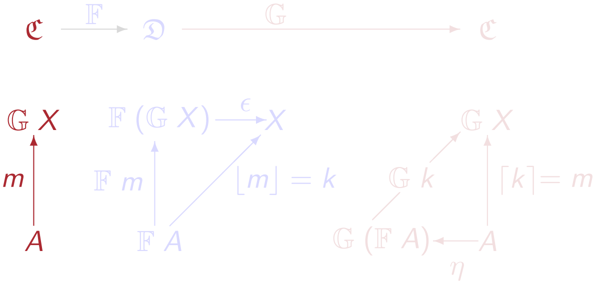
Champions league version:

$$\begin{cases} \mathbb{F}\ (\mathbb{G}\ X) \xrightarrow{\ \epsilon\ } X \\ A \xleftarrow{\ \eta\ } \mathbb{G}\ (\mathbb{F}\ A) \end{cases} \tag{7}$$

where $\mathbb{F}$ and $\mathbb{G}$ are functors.

(More about $\epsilon$ and $\eta$ later.)

# The big game

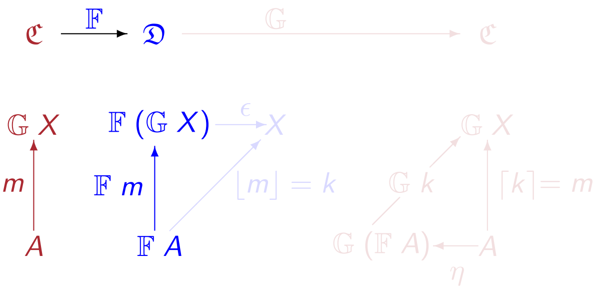



We have an **adjunction** if

$$\mathfrak{D}\ (\mathbb{F}\ A, X) \quad \cong \quad \mathfrak{C}\ (A, \mathbb{G}\ X) \tag{8}$$

and say $\mathbb{F}$ and $\mathbb{G}$ are **adjoint functors**, writing $\mathbb{F} \dashv \mathbb{G}$ as before.

# The big game

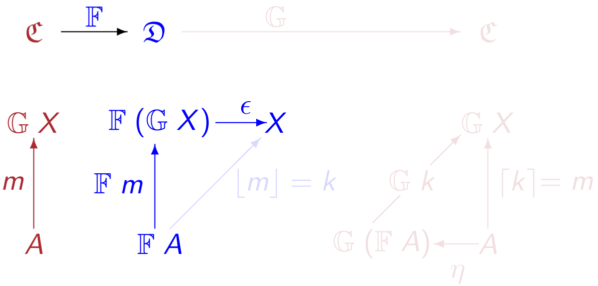



We have an **adjunction** if

$$\mathfrak{D}\ (\mathbb{F}\ A, X) \;\cong\; \mathfrak{C}\ (A, \mathbb{G}\ X) \tag{8}$$

and say $\mathbb{F}$ and $\mathbb{G}$ are **adjoint functors**, writing $\mathbb{F} \dashv \mathbb{G}$ as before.

# The big game

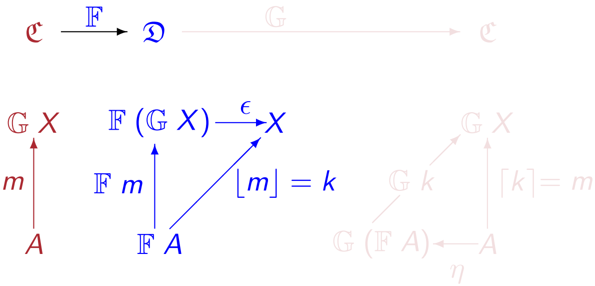



We have an **adjunction** if

$$\mathfrak{D}\,(\mathbb{F}\,A, X) \;\cong\; \mathfrak{C}\,(A, \mathbb{G}\,X) \tag{8}$$

and say $\mathbb{F}$ and $\mathbb{G}$ are **adjoint functors**, writing $\mathbb{F} \dashv \mathbb{G}$ as before.

# The big game

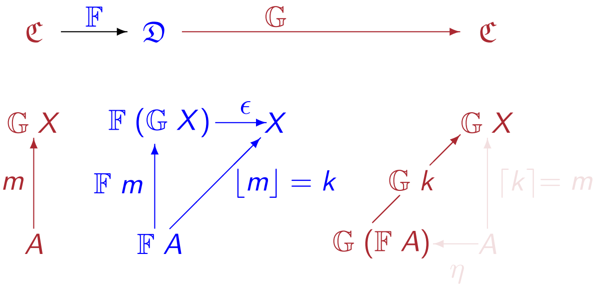



We have an **adjunction** if

$$\mathfrak{D}\ (\mathbb{F}\ A, X) \cong \mathfrak{C}\ (A, \mathbb{G}\ X) \tag{8}$$

and say $\mathbb{F}$ and $\mathbb{G}$ are **adjoint functors**, writing $\mathbb{F} \dashv \mathbb{G}$ as before.

# The big game

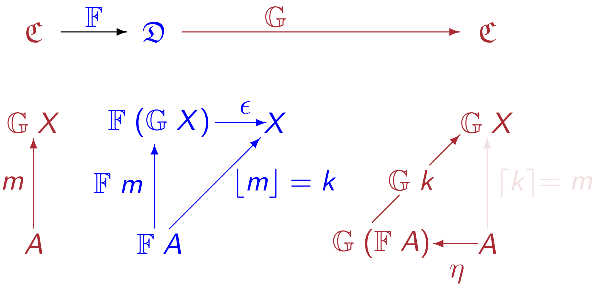



We have an **adjunction** if

$$\mathfrak{D}\ (\mathbb{F}\ A, X) \quad \cong \quad \mathfrak{C}\ (A, \mathbb{G}\ X) \tag{8}$$

and say $\mathbb{F}$ and $\mathbb{G}$ are **adjoint functors**, writing $\mathbb{F} \dashv \mathbb{G}$ as before.

# The big game



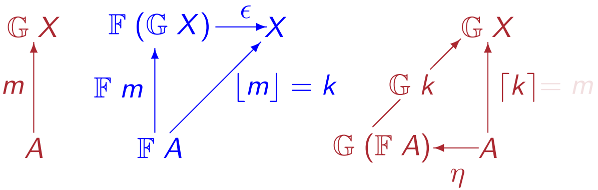

We have an **adjunction** if

$$\mathfrak{D}\ (\mathbb{F}\ A, X) \quad \cong \quad \mathfrak{C}\ (A, \mathbb{G}\ X) \tag{8}$$

and say $\mathbb{F}$ and $\mathbb{G}$ are **adjoint functors**, writing $\mathbb{F} \dashv \mathbb{G}$ as before.

# The big game

$$\mathfrak{C} \xrightarrow{\mathbb{F}} \mathfrak{D} \xrightarrow{\mathbb{G}} \mathfrak{C}$$

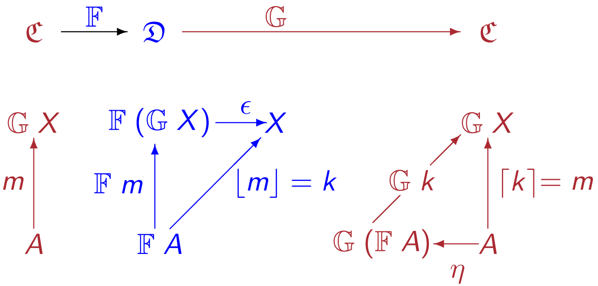



We have an **adjunction** if

$$\mathfrak{D}\ (\mathbb{F}\ A, X) \cong \mathfrak{C}\ (A, \mathbb{G}\ X) \quad (8)$$

and say $\mathbb{F}$ and $\mathbb{G}$ are **adjoint functors**, writing $\mathbb{F} \dashv \mathbb{G}$ as before.

# The big game

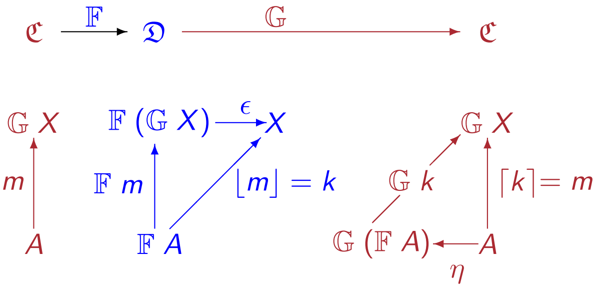



We have an **adjunction** if

$$\mathfrak{D}\ (\mathbb{F}\ A, X) \quad \cong \quad \mathfrak{C}\ (A, \mathbb{G}\ X) \tag{8}$$

and say $\mathbb{F}$ and $\mathbb{G}$ are **adjoint functors**, writing $\mathbb{F} \dashv \mathbb{G}$ as before.

# The big game




We have an **adjunction** if

$$\mathfrak{D}\ (\mathbb{F}\ A, X) \quad \cong \quad \mathfrak{C}\ (A, \mathbb{G}\ X) \tag{8}$$

and say $\mathbb{F}$ and $\mathbb{G}$ are **adjoint functors**, writing $\mathbb{F} \dashv \mathbb{G}$ as before.

# Adjunction $\mathbb{L} \dashv \mathbb{R}$

Terminology:

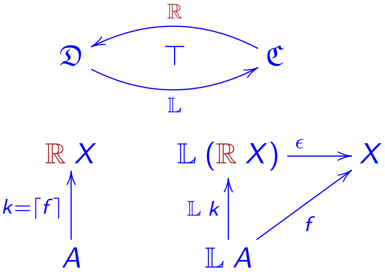
$$\begin{array}{ccc} \mathfrak{C} & & \mathfrak{D} \\ \mathbb{L}\,A \to X & \underset{\lfloor - \rfloor}{\overset{\lceil - \rceil}{\cong}} & A \to \mathbb{R}\,X \end{array} \tag{9}$$

- $\mathbb{L}$ — left adjoint
- $\mathbb{R}$ — right adjoint
- $\lceil f \rceil$ — $\mathbb{R}$-transpose of $f$
- $\lfloor g \rfloor$ — $\mathbb{L}$-transpose of $g$

# Adjunction $\mathbb{L} \dashv \mathbb{R}$

In detail — **universal property**:

$$k = \lceil f \rceil \;\Leftrightarrow\; \underbrace{\epsilon \cdot \mathbb{L}\, k}_{\lfloor k \rfloor} = f$$




Terminology — $\epsilon = \lfloor id \rfloor$ is called the **co-unit** of the adjunction.

# (Covariant) exponentials: $(\_ \times K) \dashv (\_^K)$

Perhaps the most famous adjunction:

$$\begin{cases} \mathbb{L}\, X = X \times K \\ \mathbb{R}\, X = X^K \\ \epsilon = \mathbf{ev} \end{cases} \qquad \begin{cases} \lceil f \rceil = \mathbf{curry}\, f \\ \lfloor f \rfloor = \mathbf{uncurry}\ f \end{cases}$$

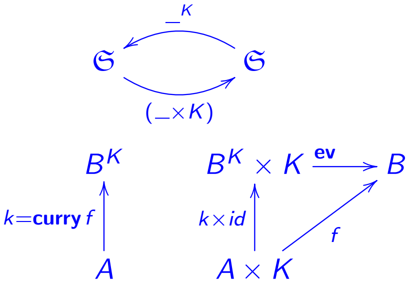
$$A \times K \to X \underset{\mathbf{uncurry}}{\overset{\mathbf{curry}}{\cong}} A \to X^K$$

where

$$\mathbf{curry}\, f\ a\ b = f\ (a, b)$$
$$\mathbf{uncurry}\ g\ (a, b) = g\ a\ b$$
$$\mathbf{ev}\ (f, k) = f\ k$$

# (Covariant) exponentials: $(\_ \times K) \dashv (\_^K)$

$$k = \textbf{curry}\, f \;\Leftrightarrow\; \underbrace{\textbf{ev} \cdot (k \times id)}_{\textbf{uncurry}\ k} = f$$

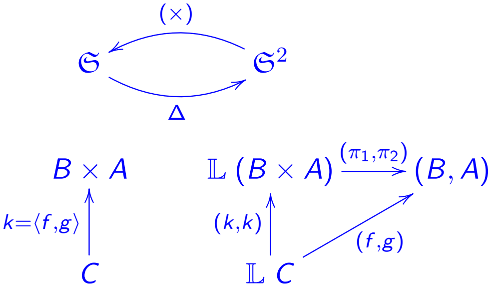



$$\textbf{Functor} : \qquad f^K = (f \cdot) \tag{10}$$

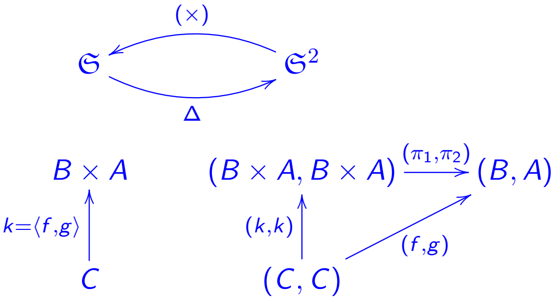
# Pairing: $\Delta \dashv \times$

$$\begin{cases} \mathbb{L}\ X = \Delta\ X = (X, X) \\ \mathbb{R}\ (X, Y) = X \times Y \\ \epsilon = (\pi_1, \pi_2) \end{cases}$$

$$\begin{cases} \lceil (f, g) \rceil = \langle f, g \rangle \\ \lfloor k \rfloor = (\pi_1 \cdot k, \pi_2 \cdot k) \end{cases}$$



# Pairing: $\Delta \dashv \times$



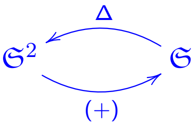

That is:

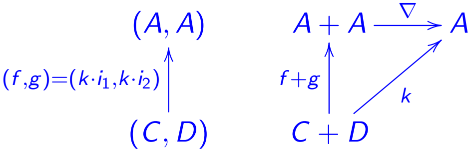
$$k = \langle f, g \rangle \quad \Leftrightarrow \quad \begin{cases} \pi_1 \cdot k = f \\ \pi_2 \cdot k = g \end{cases} \tag{11}$$

# Co-pairing: $(+) \dashv \Delta$

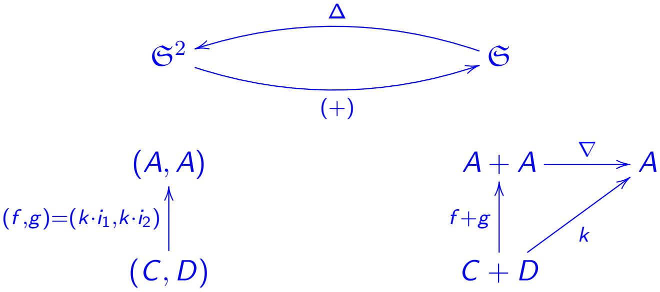
$$\begin{cases} \mathbb{L}\,(X,Y) = X + Y \\ \mathbb{R}\,X = \Delta\,X = (X,X) \\ \epsilon = \nabla = [id, id] \end{cases}$$

$$\begin{cases} \lceil k \rceil = (k \cdot i_1, k \cdot i_2) \\ \lfloor (f,g) \rfloor = [f,g] \end{cases}$$



# Co-pairing: $+ \dashv \Delta$



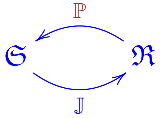

$$\begin{cases} f = k \cdot i_1 \\ g = k \cdot i_2 \end{cases} \quad \Leftrightarrow \quad k = [f, g]$$

# Power transpose: $\mathbb{J} \dashv \mathbb{P}$

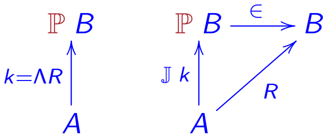
$\mathfrak{D} := \mathfrak{S}$ (sets + functions) and $\mathfrak{C} := \mathfrak{R}$ (sets + relations)

$$\begin{cases} \mathbb{J}\ X = X \\ y\ (\mathbb{J}\ k)\ x \Leftrightarrow y = k\ x \end{cases} \qquad \begin{cases} \lceil R \rceil = \Lambda R \\ y\ \lfloor k \rfloor\ x = y \in (k\ x) \end{cases}$$

$\in : A \leftarrow \mathbb{P}\ A$ is the set **membership** relation




$$k = \Lambda R \;\Leftrightarrow\; \underbrace{\in \cdot\, k}_{\lfloor k \rfloor} = R$$



# Corollaries of $k = \lceil f \rceil \Leftrightarrow \epsilon \cdot \mathbb{L}\, k = f$

*reflection*:

$$\lceil \epsilon \rceil = id \tag{12}$$

that is,

$$\epsilon = \lfloor id \rfloor \tag{13}$$

*cancellation*:

$$\epsilon \cdot \mathbb{L}\, \lceil f \rceil = f \tag{14}$$

*fusion*:

$$\lceil h \rceil \cdot g = \lceil h \cdot \mathbb{L}\, g \rceil \tag{15}$$

# Corollaries

*absorption*:

$$(\mathbb{R}\ g) \cdot \lceil h \rceil = \lceil g \cdot h \rceil \tag{16}$$

*naturality*:

$$h \cdot \epsilon = \epsilon \cdot \mathbb{L}\ (\mathbb{R}\ h) \tag{17}$$

*closed definition*:

$$\lfloor k \rfloor = \epsilon \cdot (\mathbb{L}\ k) \tag{18}$$

*functor*

$$\mathbb{R}\ h = \lceil h \cdot \epsilon \rceil \tag{19}$$

# Dual formulation

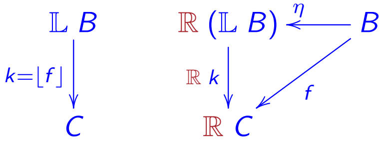
As with **GC**s, universal property can be expressed in a dual way, as follows:

$$
\begin{array}{ll}
 & k = \lfloor f \rfloor \\
\Leftrightarrow & \quad \{ \text{ identity; homset isomorphism } \} \\
 & \lceil k \cdot id \rceil = f \\
\Leftrightarrow & \quad \{ \text{ absorption (16) ; } \lceil id \rceil = \eta \ \} \\
 & \underbrace{(\mathbb{R}\ k) \cdot \eta}_{\lceil k \rceil} = f
\end{array}
$$

# Dual formulation

Diagram:

$$k = \lfloor f \rfloor \;\Leftrightarrow\; \underbrace{\mathbb{R}\; k \cdot \eta}_{\lceil k \rceil} = f$$




---

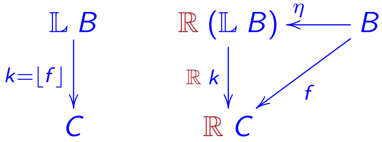
Example ($\mathbb{J} \dashv \mathbb{P}$): $\mathfrak{R}$ $\mathfrak{S}$

$$R = \underbrace{\in \cdot f}_{\lfloor f \rfloor} \;\Leftrightarrow\; \underbrace{\mathbb{P}\; R \cdot \eta}_{\Lambda R} = f$$

# Dual formulation

Diagram:

$$k = \lfloor f \rfloor \;\Leftrightarrow\; \underbrace{\mathbb{R}\,k \cdot \eta}_{\lceil k \rceil} = f$$

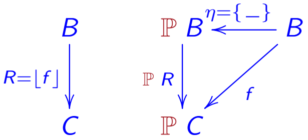



---

Example ($\mathbb{J} \dashv \mathbb{P}$):

$\mathfrak{R}$ $\mathfrak{S}$

$$R = \underbrace{\in \cdot f}_{\lfloor f \rfloor} \;\Leftrightarrow\; \underbrace{\mathbb{P}\,R \cdot \eta}_{\Lambda R} = f$$



# Dual corollaries

Now arising from $k = \lfloor f \rfloor \Leftrightarrow \underbrace{\mathbb{R}\ k \cdot \eta}_{\lceil k \rceil} = f$

*reflection*:

$$\lfloor \eta \rfloor = id \tag{20}$$

that is,

$$\eta = \lceil id \rceil \tag{21}$$

*cancellation*:

$$\mathbb{R}\ \lfloor f \rfloor \cdot \eta = f \tag{22}$$

*fusion*:

$$g \cdot \lfloor h \rfloor = \lfloor \mathbb{R}\ g \cdot h \rfloor \tag{23}$$

# Dual corollaries

*absorption*:

$$\lfloor h \rfloor \cdot \mathbb{L}\, g = \lfloor h \cdot g \rfloor \tag{24}$$

*naturality*:

$$h \cdot \epsilon = \epsilon \cdot \mathbb{L}\,(\mathbb{R}\ h) \tag{25}$$

*closed definition*:

$$\lceil g \rceil = (\mathbb{R}\ g) \cdot \eta \tag{26}$$

*functor*

$$\mathbb{L}\, g = \lfloor \eta \cdot g \rfloor \tag{27}$$

*cancellation (corollary)*:

$$\epsilon \cdot \mathbb{L}\,\eta = id \tag{28}$$

# Adjunction composition (exchange law)

Assuming $\mathbb{L} \dashv \mathbb{M} \dashv \mathbb{R}$ and inspecting $\mathbb{L}\ A \xrightarrow{k} \mathbb{R}\ B$ :

$$
\begin{array}{ll}
 & \mathbb{M}\ \mathbb{L}\ A \to B \\
\cong & \qquad \{\ \mathbb{M} \dashv \mathbb{R}\ \} \\
 & \mathbb{L}\ A \to \mathbb{R}\ B \\
\cong & \qquad \{\ \mathbb{L} \dashv \mathbb{M}\ \} \\
 & A \to \mathbb{M}\ \mathbb{R}\ B
\end{array}
$$

On the one hand, $k = \lceil f \rceil_{\mathbb{R}}$ for exactly one $\mathbb{M}\ \mathbb{L}\ A \xrightarrow{f} B$ .

On the other hand, $k = \lfloor g \rfloor_{\mathbb{L}}$ for exactly one $A \xrightarrow{g} \mathbb{M}\ \mathbb{R}\ B$ .

So the **exchange law**

$$\lceil f \rceil_{\mathbb{R}} = \lfloor g \rfloor_{\mathbb{L}} \tag{29}$$

holds for such $\mathbb{M}\ \mathbb{L}\ A \xrightarrow{f} B$ and $A \xrightarrow{g} \mathbb{M}\ \mathbb{R}\ B$ .

$$(+) \dashv \Delta \dashv (\times)$$

$\mathbb{M}\ \mathbb{L}\ A \xrightarrow{f} B$ is of type $\Delta\ (+)\ (A, C) \longrightarrow (B, D)$ :

$$f = (A + C, A + C) \xrightarrow{(m,n)} (B, D)$$

$A \xrightarrow{g} \mathbb{M}\ \mathbb{R}\ B$ is of type $(A, C) \longrightarrow \Delta\ (\times)\ (B, D)$ :

$$g = (A, C) \xrightarrow{(i,j)} (B \times D, B \times D)$$

So

$$\lceil f \rceil_{\mathbb{R}} = \lfloor g \rfloor_{\mathbb{L}} \quad \textit{becomes} \quad \langle m, n \rangle = [i, j]$$

$$(+) \dashv \Delta \dashv (\times)$$

$\mathbb{M}\,\mathbb{L}\,A \xrightarrow{f} B$ is of type $\Delta\,(+)\,(A, C) \longrightarrow (B, D)$ :

$$f = \; (A + C, A + C) \xrightarrow{(m,n)} (B, D)$$

$A \xrightarrow{g} \mathbb{M}\,\mathbb{R}\,B$ is of type $(A, C) \longrightarrow \Delta\,(\times)\,(B, D)$ :

$$g = \; (A, C) \xrightarrow{(i,j)} (B \times D, B \times D)$$

So

$\lceil f \rceil_{\mathbb{R}} = \lfloor g \rfloor_{\mathbb{L}}$ *becomes* $\langle m, n \rangle = [i, j]$

# Solving $\langle m, n\rangle = [i, j]$

Find $m$ and $n$ for $i = \langle h, k\rangle$ and $j = \langle p, q\rangle$ in:

$$\langle m, n\rangle = [\langle h, k\rangle, \langle p, q\rangle]$$

$$\Leftrightarrow \quad \{ \ (+) \dashv \Delta \ \}$$

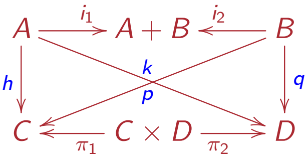
$$\begin{cases} (m, n)\ (i_1, i_1) = (h, k) \\ (m, n)\ (i_2, i_2) = (p, q) \end{cases}$$

$$\Leftrightarrow \quad \{ \ \text{re-arranging} \ \}$$

$$\begin{cases} (m, m)\ (i_1, i_2) = (h, p) \\ (n, n)\ (i_1, i_2) = (k, q) \end{cases}$$

$$\Leftrightarrow \quad \{ \ \Delta \dashv (\times) \ \}$$

$$\begin{cases} m = [h, p] \\ n = [k, q] \end{cases}$$

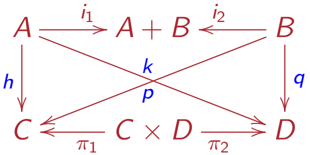


## $(+) \dashv \Delta \dashv (\times)$

The composition of the two adjunctions therefore yields the

**exchange law**:

$$\langle [h, p], [k, q] \rangle = [\langle h, k \rangle, \langle p, q \rangle] \quad (30)$$

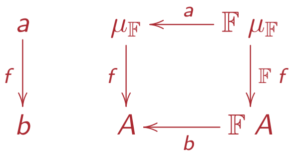



(As will be seen later, this law will play a role when dealing with mutual recursion.)

# Recursion comes in

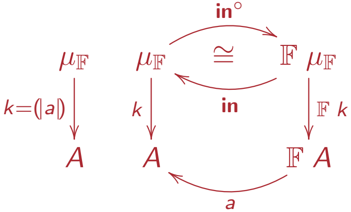
**Algebras** $A \xleftarrow{a} \mathbb{F}\, A$

**Morphisms** $a \xrightarrow{f} b$
between $\mathbb{F}$-algebras




lead to $\mathbb{F}$-recursion.

Initial algebra $\mu_{\mathbb{F}} \xleftarrow{\mathbf{in}} \mathbb{F}\, \mu_{\mathbb{F}}$ such that morphism $\mathbf{in} \xrightarrow{(\!|a|\!)} a$ is unique:




Universal property:

$$k = (\!|a|\!) \Leftrightarrow k \cdot \mathbf{in} = a \cdot \mathbb{F}\, k \quad (31)$$

Terminology: $(\!|\_|\!)$ = **catamorphism**.

# $(\!|\_|\!)$ meets $\mathbb{L} \dashv \mathbb{R}$

Chemistry with recursion:

$$\lceil f \rceil = (\!| \lceil h \rceil |\!)$$

$$\Leftrightarrow \quad \{ \text{ cata-universal (31) } \}$$

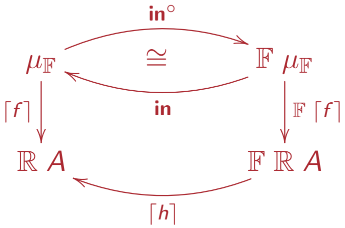
$$\lceil f \rceil \cdot \mathbf{in} = \lceil h \rceil \cdot \mathbb{F} \lceil f \rceil$$

$$\Leftrightarrow \quad \{ \text{ fusion (15) twice } \}$$

$$\lceil f \cdot \mathbb{L}\, \mathbf{in} \rceil = \lceil h \cdot \mathbb{L}\, \mathbb{F} \lceil f \rceil \rceil$$

$$\Leftrightarrow \quad \{ \text{ isomorphism } \lceil \_ \rceil \}$$

$$f \cdot \mathbb{L}\, \mathbf{in} = h \cdot \mathbb{L}\, \mathbb{F} \lceil f \rceil$$

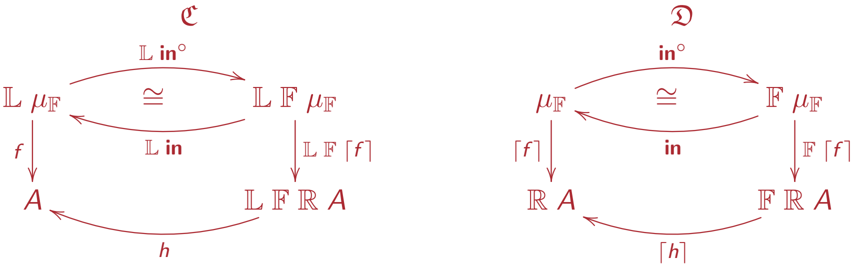


# $(\!|\_|\!)$ meets $\mathbb{L} \dashv \mathbb{R}$

Therefore:

$$f \cdot \mathbb{L}\,\mathbf{in} = h \cdot \mathbb{L}\,\mathbb{F}\,\lceil f \rceil \quad \Leftrightarrow \quad \lceil f \rceil = (\!|\lceil h \rceil|\!) \tag{32}$$

Diagrams:



# Example: $(\!|\_|\!)$ meets $\Delta \dashv (\times)$

Pairing adjunction:

$$\mathbb{L}\, f = \Delta\, f = (f, f)$$
$$\epsilon = (\pi_1, \pi_2)$$
$$\lceil (f, g) \rceil = \langle f, g \rangle$$

Left-hand side:

$$
\begin{array}{ll}
 & (f, g) \cdot \mathbb{L}\, \mathbf{in} = (h, k) \cdot \mathbb{L}\, (\mathbb{F}\, \lceil (f, g) \rceil) \\
\Leftrightarrow & \quad \{\ \mathbb{L}\, f = (f, f)\ ;\ \lceil (f, g) \rceil = \langle f, g \rangle\ \} \\
 & (f, g) \cdot (\mathbf{in}, \mathbf{in}) = (h, k) \cdot (\mathbb{F}\, \langle f, g \rangle, \mathbb{F}\, \langle f, g \rangle) \\
\Leftrightarrow & \quad \{\ \text{composition and equality of pairs of functions}\ \} \\
 & \left\{ \begin{array}{l} f \cdot \mathbf{in} = h \cdot \mathbb{F}\, \langle f, g \rangle \\ g \cdot \mathbf{in} = k \cdot \mathbb{F}\, \langle f, g \rangle \end{array} \right.
\end{array}
$$

# Cata meets $\Delta \dashv (\times)$

Right-hand side:

$$\lceil (f, g) \rceil = (\!| \lceil (h, k) \rceil |\!)$$

$$\Leftrightarrow \qquad \{ \ \lceil (f, g) \rceil = \langle f, g \rangle \text{ twice } \}$$

$$\langle f, g \rangle = (\!| \langle h, k \rangle |\!)$$

Putting both sides together we get the **mutual recursion** law:

$$\langle f, g \rangle = (\!| \langle h, k \rangle |\!) \quad \Leftrightarrow \quad \left\{ \begin{array}{l} f \cdot \mathbf{in} = h \cdot \mathbb{F} \langle f, g \rangle \\ g \cdot \mathbf{in} = k \cdot \mathbb{F} \langle f, g \rangle \end{array} \right. \tag{33}$$

# Why mutual recursion matters

**Mutual recursion** very useful.

It comes handy in particular **dynamic programming** situations.

Examples follow in the Peano-recursion ($\mathbf{in} = [zero, succ]$) setting, whose catamorphisms (folds) are **for**-loops,

$$\mathbf{for}\ f\ i = (\![\underline{i}, f]\!)$$

that is

$$\mathbf{for}\ f\ i\ 0 = i$$
$$\mathbf{for}\ f\ i\ (n+1) = f\ (\mathbf{for}\ f\ i\ n)$$

Example (Church numerals): $church\ n\ f\ b = \mathbf{for}\ f\ b\ n$.

# Why mutual recursion matters — Fibonacci

Classic **DP** problem

$$\begin{aligned} &\mathit{fib}\ 0 = 1 \\ &\mathit{fib}\ 1 = 1 \\ &\mathit{fib}\ (n+2) = \mathit{fib}\ (n+1) + \mathit{fib}\ n \end{aligned}$$

unfolds to:

$$\begin{array}{rcl} f\ 0 &=& 1 \\ f(n+1) &=& f\ n + \mathit{fib}\ n \\ \mathit{fib}\ 0 &=& 1 \\ \mathit{fib}(n+1) &=& f\ n \end{array}$$

That is:

$$\begin{aligned} f \cdot [\mathit{zero}, \mathit{succ}] = [\underline{1}, \mathit{add}] \cdot \langle f, \mathit{fib} \rangle \\ \mathit{fib} \cdot [\mathit{zero}, \mathit{succ}] = [\underline{1}, \pi_1] \cdot \langle f, \mathit{fib} \rangle \end{aligned}$$

# Why mutual recursion matters — Fibonacci

This together with the **exchange law** (30) leads to:

$$\langle f, fib \rangle \quad = \quad (\![ [\underline{(1,1)}, \langle add, \pi_1 \rangle] ]\!) \tag{34}$$

That is (Haskell):

$$fib = snd \cdot \textbf{for } loop\ (1,1) \textbf{ where}$$
$$\quad loop\ (x,y) = (x+y, x)$$

For non-functional programmers:

```
int fib(int n)
{
   int x=1; int y=1; int i;
   for (i=1;i<=n;i++) {int a=x; x=x+y; y=a;}
   return y;
};
```

# Why mutual recursion matters — Fibonacci

This together with the **exchange law** (30) leads to:

$$\langle f, fib \rangle \quad = \quad (\![\underline{(1,1)}, \langle add, \pi_1 \rangle]\!) \qquad (34)$$

That is (Haskell):

$fib = snd \cdot$ **for** $loop\ (1,1)$ **where**
$\quad loop\ (x,y) = (x+y, x)$

For non-functional programmers:

```
int fib(int n)
{
   int x=1; int y=1; int i;
   for (i=1;i<=n;i++) {int a=x; x=x+y; y=a;}
   return y;
};
```

# Why mutual recursion matters — Catalan numbers

$$C_n = \frac{(2n)!}{(n+1)!(n!)}$$

Lots of factorial (re)calculations — try "**DP** artilhery"?

No — use **mutual recursion** instead, based on this property:

$$C_{n+1} = \frac{4n+2}{n+2} C_n$$

Three functions in mutual recursion:

$$c\ n = C_n$$
$$f\ n = 4n + 2$$
$$g\ n = n + 2$$

Then (next slide):

# Why mutual recursion matters — Catalan numbers

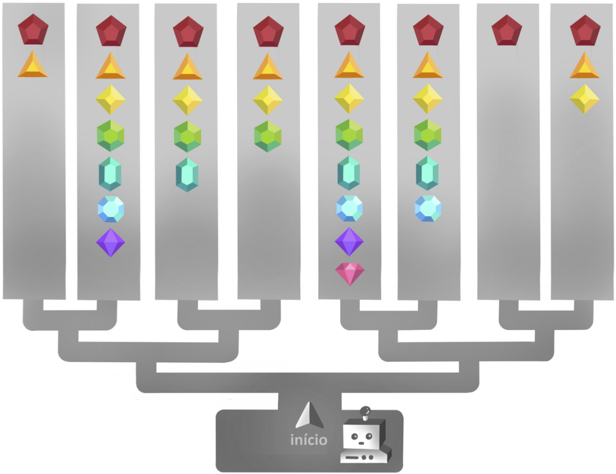
"Peano unfolding":

$$
\begin{aligned}
c\ 0 &= 1 \\
c\ (n+1) &= \frac{(f\ \ n) \times (c\ \ n)}{g\ \ n} \\
f\ 0 &= 2 \\
f\ (n+1) &= f\ \ n + 4 \\
g\ 0 &= 2 \\
g\ (n+1) &= g\ \ n + 1
\end{aligned}
$$

Finally applying the law we get a **for**-loop with 3 local variables:

$$
\begin{array}{l}
c = prj \cdot (\mathbf{for}\ loop\ init)\ \mathbf{where} \\
\quad loop\ (c, f, g) = ((f * c) \div g, f + 4, g + 1) \\
\quad inic = (1, 2, 2) \\
\quad prj\ (c, \_, \_) = c
\end{array}
$$

# Why mutual recursion matters — minimax



# Why mutual recursion matters — minimax

Wikipedia:

```
function  minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

```
(* Initial call *)
minimax(origin, depth, TRUE)
```

# Why mutual recursion matters — minimax

Mutual recursion (players *alice* and *bob*):

$$minimax = \langle alice, bob \rangle$$

where

$$\begin{cases} alice \cdot \mathbf{in} = [id, umax] \cdot \mathbb{F}\ bob \\ bob \cdot \mathbf{in} = [id, umin] \cdot \mathbb{F}\ alice \end{cases}$$

assuming

$$\mathbf{in} = [Leaf, Fork]$$

$$\mathbb{F}\ f = id + f \times f$$

in the contex of

$$\mathbf{data}\ LTree\ a = Leaf\ \ a \mid Fork\ (LTree\ a, LTree\ a)$$

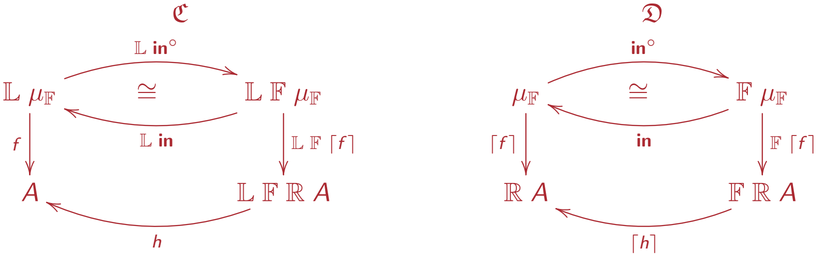
(generalizable to other $\mathbb{F}$ tree-structures).

# Further chemistry with recursion

Back to (32), recall

$$f \cdot \mathbb{L}\,\mathbf{in} = h \cdot \mathbb{L}\,\mathbb{F}\,\lceil f \rceil \quad \Leftrightarrow \quad \lceil f \rceil = (\!|\lceil h \rceil|\!)$$

and the diagram:




How to get $f$ instead of $\lceil f \rceil$ in the recursive call to obtain $f$ as a **hylomorphism**?

# Further chemistry with recursion

The resource we have for this is **cancellation** (14):

$$\epsilon \cdot \mathbb{L} \lceil f \rceil = f$$

However, $\mathbb{L}$ in $\mathbb{L}\ \mathbb{F} \lceil f \rceil$ is in the wrong position and needs to commute with $\mathbb{F}$.

We need a **distributive** law $\mathbb{L}\ \mathbb{F} \to \mathbb{F}\ \mathbb{L}$.

More generally, we rely on some **natural transformation**

$$\phi : \mathbb{L}\ \mathbb{F} \to \mathbb{G}\ \mathbb{L}$$

enabling such a commutation over some $\mathbb{G}$.

# Further chemistry with recursion

For $\epsilon \cdot \mathbb{L} \lceil f \rceil = f$ to be of use, we need $\mathbb{G}\ \epsilon$ somewhere in the pipeline.

We thus refine $h := h \cdot \mathbb{G}\ \epsilon \cdot \phi$ above and carry on:

$$\lceil f \rceil = (\!|\lceil h \cdot \mathbb{G}\ \epsilon \cdot \phi \rceil|\!)$$

$$\Leftrightarrow \qquad \{ \ (32) \ \}$$

$$f \cdot \mathbb{L}\ \mathbf{in} = h \cdot \mathbb{G}\ \epsilon \cdot \phi \cdot \mathbb{L}\ \mathbb{F}\ \lceil f \rceil$$

$$\Leftrightarrow \qquad \{ \ \text{natural-}\phi\text{:}\ \phi \cdot \mathbb{L}\ \mathbb{F}\ f = \mathbb{G}\ \mathbb{L}\ f \cdot \phi \ \}$$

$$f \cdot \mathbb{L}\ \mathbf{in} = h \cdot \mathbb{G}\ \epsilon \cdot \mathbb{G}\ \mathbb{L}\ \lceil f \rceil \cdot \phi$$

$$\Leftrightarrow \qquad \{ \ \text{functor}\ \mathbb{G}\text{; cancellation}\ \epsilon \cdot \mathbb{L}\ \lceil f \rceil = f\ (14) \ \}$$

$$f \cdot \mathbb{L}\ \mathbf{in} = h \cdot \mathbb{G}\ f \cdot \phi$$

# $\mathbb{G}$-hylo adjoint to $\mathbb{F}$-cata

We reach

$$\underbrace{f \cdot (\mathbb{L}\ \mathbf{in}) = h \cdot \mathbb{G}\ f \cdot \phi}_{\mathbb{G}\text{-hylomorphism}} \quad \Leftrightarrow \quad \underbrace{\lceil f \rceil = (\!|\lceil h \cdot \mathbb{G}\ \epsilon \cdot \phi \rceil|\!)}_{\text{adjoint } \mathbb{F}\text{-catamorphism}} \tag{35}$$

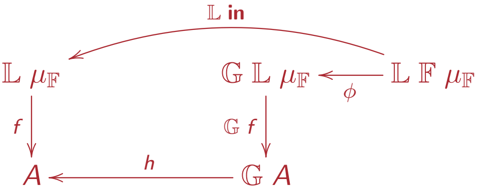
where natural transformation

$$\phi : \mathbb{L}\ \mathbb{F} \rightarrow \mathbb{G}\ \mathbb{F}$$

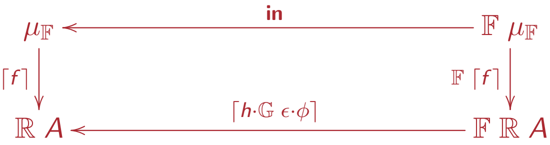
captures the necessary switch of recursion-pattern between **hylo** ($\mathbb{G}$) and **cata** ($\mathbb{F}$).

# Diagrams

$\mathbb{G}$-hylo ($\mathfrak{C}$):




Adjoint $\mathbb{F}$-cata ($\mathfrak{D}$):




$$A \xleftarrow{h} \mathbb{G}\ A \xleftarrow{\mathbb{G}\ \epsilon} \mathbb{G}\ \mathbb{L}\ \mathbb{R}\ A \xleftarrow{\phi} \mathbb{L}\ \mathbb{F}\ \mathbb{R}\ A$$

# $\mathbb{G}$-hylo-universal

The interest in

$$f \cdot (\mathbb{L}\ \mathbf{in}) = h \cdot \mathbb{G}\ f \cdot \phi \quad \Leftrightarrow \quad \lceil f \rceil = (\!|\lceil h \cdot \mathbb{G}\ \epsilon \cdot \phi \rceil|\!)$$

is that one can use "**cata**-artilhery" to reason about **hylo** $f$.

But not necessarily: $\lfloor \_ \rfloor$-"shunting" on the right side

$$\underbrace{f \cdot (\mathbb{L}\ \mathbf{in}) = h \cdot \mathbb{G}\ f \cdot \phi}_{\mathbb{G}\text{-hylomorphism}} \quad \Leftrightarrow \quad f = \underbrace{\lfloor (\!|\lceil h \cdot \mathbb{G}\ \epsilon \cdot \phi \rceil|\!) \rfloor}_{\langle\!| h |\!\rangle}$$

gives us a new combinator with **universal property**:

$$f = \langle\!| h |\!\rangle \quad \Leftrightarrow \quad f \cdot \mathbb{L}\ \mathbf{in} = h \cdot \mathbb{G}\ f \cdot \phi \qquad (36)$$

# $(\!|\_|\!)$ fusion, reflection and so on

*fusion*:

$$k \cdot (\!| f |\!) = (\!| g |\!) \quad \Leftarrow \quad k \cdot f = g \cdot \mathbb{G}\, k \tag{37}$$

*reflection* (in case $\phi$ is an **isomorphism**):

$$(\!| \alpha |\!) = id \tag{38}$$

where $\alpha$ abbreviates $\mathbb{L}\, \mathbf{in} \cdot \phi^{\circ}$ in

$$f = (\!| h |\!) \quad \Leftrightarrow \quad f \cdot \underbrace{\mathbb{L}\, \mathbf{in} \cdot \phi^{\circ}}_{\alpha} = h \cdot \mathbb{G}\, f$$

*cancellation*:

$$(\!| h |\!) \cdot \alpha = h \cdot \mathbb{G}\, (\!| h |\!)$$

# Many applications!

Many results in the literature arise as instances of this theorem.

For instance, the **structural recursion theorem** of Bird and de Moor (1997):

**Theorem 3.1** If $\phi$ is natural in the sense that $\mathsf{G}(h \times id) \cdot \phi = \phi \cdot (\mathsf{F}h \times id)$, then

$$f \cdot (\alpha \times id) \quad = \quad h \cdot \mathsf{G}f \cdot \phi$$

if and only if

$$f \quad = \quad apply \cdot (([curry\,(h \cdot \mathsf{G}apply \cdot \phi)]) \times id).$$

Details:

$$\mathbb{L} \dashv \mathbb{R} := (\times K) \dashv (\_^K)) \qquad \begin{cases} \mathbb{F}\, X = 1 + A \times X \\ \mathbb{G}\, X = (1 + K) + A \times X \end{cases}$$

$$\phi = (id + \mathsf{assocr}) \cdot distl$$

# Many applications!

Many results in the literature arise as instances of this theorem. For instance, the **structural recursion theorem** of Bird and de Moor (1997):

**Theorem 3.1** If $\phi$ is natural in the sense that $\mathsf{G}(h \times id) \cdot \phi = \phi \cdot (\mathsf{F}h \times id)$, then

$$f \cdot (\alpha \times id) \quad = \quad h \cdot \mathsf{G}f \cdot \phi$$

if and only if

$$f \quad = \quad apply \cdot (⦇curry\,(h \cdot \mathsf{G}apply \cdot \phi)⦈ \times id).$$

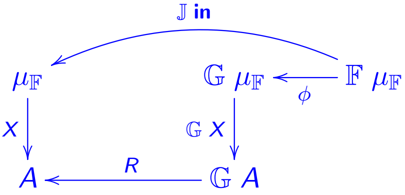
Details:

$$\mathbb{L} \dashv \mathbb{R} := (\times K) \dashv (\_^K)) \qquad \begin{cases} \mathbb{F}\,X = 1 + A \times X \\ \mathbb{G}\,X = (1 + K) + A \times X \end{cases}$$

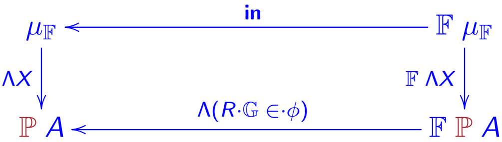
$$\phi = (id + \mathsf{assocr}) \cdot distl$$

# Relational catas thanks to $\mathbb{J} \dashv \mathbb{P}$

$\mathbb{G}$-hylo (**relational**):




Adjoint $\mathbb{F}$-cata (**functional**):



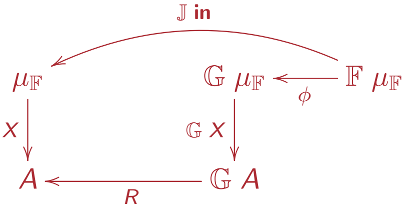

$$A \xleftarrow{R} \mathbb{G}\ A \xleftarrow{\mathbb{G}\ \in} \mathbb{G}\ \mathbb{P}\ A \xleftarrow{\phi} \mathbb{F}\ \mathbb{P}\ A$$

# Relational catas thanks to $\mathbb{J} \dashv \mathbb{P}$

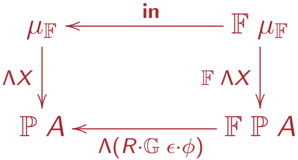
Recall (relational side):

$$\begin{cases} \mathbb{J}\ X = \mathbb{J}\ X \\ y\ (\mathbb{J}\ f)\ x \Leftrightarrow y = f\ x \end{cases}$$

relational $\mathbb{G}$-recursion




functional $\mathbb{F}$-recursion



# Relational catas thanks to $\mathbb{J} \dashv \mathbb{P}$

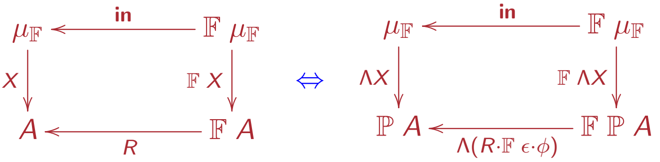
Recall (relational side):

$$\begin{cases} \mathbb{J}\ X = X \\ y\ (\mathbb{J}\ f)\ x \Leftrightarrow y = f\ x \end{cases}$$

Because $\mathbb{J}\ X = X$ we can choose $\mathbb{G}\ X = \mathbb{F}\ X$ and $\phi = id$.

**Functor** $\mathbb{F}$ extends to a **relator** $\mathbb{G}$.

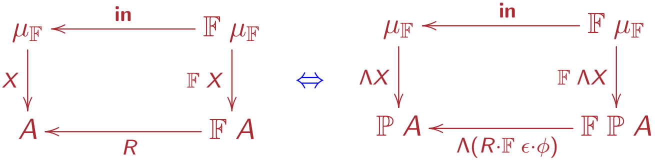
As is usual, we use the same symbol for functor and relator, greatly simplifying diagrams:



# Relational catas thanks to $\mathbb{J} \dashv \mathbb{P}$

$\lfloor \_ \rfloor$-shunting again:

$$X \cdot \mathbf{in} = R \cdot \mathbb{F}\, X \quad \Leftrightarrow \quad \Lambda X = (\!|\Lambda(R \cdot \mathbb{F} \in)|\!)$$




$$X \cdot \mathbf{in} = R \cdot \mathbb{F}\, X \quad \Leftrightarrow \quad X = \underbrace{\in \cdot\, (\!|\Lambda(R \cdot \mathbb{F} \in)|\!)}_{(\!|R|\!)} \tag{39}$$

This extends "banana-brackets" to **relations** and gives birth to **inductive relations**.

# Eilenberg-Wright Lemma

Put in another way:

The equivalence

$$X = (\!|R|\!) \quad \Leftrightarrow \quad \Lambda X = (\!|\Lambda(R \cdot \mathbb{F} \in)|\!) \tag{40}$$

— known as the **Eilenberg-Wright** Lemma —

follows from the "adjoint catamorphism" theorem (35)[1] for the **power-transpose** adjunction $\mathbb{J} \dashv \mathbb{P}$.

[1] Also known as "adjoint fold" theorem (Hinze, 2013).

# Relational catas thanks to $\mathbb{J} \dashv \mathbb{P}$

In summary,

$$(\mathbb{J} \dashv \mathbb{P}) + (\!|\_|\!)$$

leads as to **inductive relations**, with universal property:

$$X \cdot \mathbf{in} = R \cdot \mathbb{F}\, X \quad \Leftrightarrow \quad X = (\!|R|\!)$$

Instance for **Peano recursion**, where

$$\mathbf{in} = [zero, succ]$$
$$\mathbb{F}\, X = id + X$$

but this time relationally:

$$X = (\!|R|\!) \quad \Leftrightarrow \quad \begin{cases} X \cdot zero = R \cdot i_1 \\ X \cdot succ = R \cdot i_2 \cdot X \end{cases}$$

# Relational catas thanks to $\mathbb{J} \dashv \mathbb{P}$

In summary,

$$(\mathbb{J} \dashv \mathbb{P}) + (\!|\_|\!)$$

leads as to **inductive relations**, with universal property:

$$X \cdot \mathbf{in} = R \cdot \mathbb{F}\, X \quad \Leftrightarrow \quad X = (\!| R |\!)$$

Instance for **Peano recursion**, where

$$\mathbf{in} = [zero, succ]$$
$$\mathbb{F}\, X = id + X$$

but this time relationally:

$$X = (\!| R |\!) \quad \Leftrightarrow \quad \left\{ \begin{array}{l} X \cdot zero = R \cdot i_1 \\ X \cdot succ = R \cdot i_2 \cdot X \end{array} \right.$$

# Inductive relations thanks to $\mathbb{J} \dashv \mathbb{P}$

Remember $\mathbf{N}_0 \xleftarrow{(\geqslant)} \mathbf{N}_0$ ?

Now we know how to define it over the Peano algebra,

$$(\geqslant) = (\![\top, succ]\!]) \tag{41}$$

where $\top$ is the largest relation of its type. ($b \top a = True$ for all $a$ and $b$.)
Unfolding (41):

$$(\geqslant) = (\![\top, succ]\!])$$

$\Leftrightarrow$ { previous slide }

$$\begin{cases} (\geqslant) \cdot zero = \top \\ (\geqslant) \cdot succ = succ \cdot (\geqslant) \end{cases}$$

$\Leftrightarrow$ { go pointwise (in $\mathfrak{R}$) }

$$\begin{cases} y \geqslant 0 = True \\ y \geqslant (x+1) = \langle \exists\ z\ :\ y = z+1 :\ z \geqslant x \rangle \end{cases}$$

$\square$

# Inductive relations thanks to $\mathbb{J} \dashv \mathbb{P}$

Remember list **prefixes** and **sublists**, $ys \sqsubseteq xs$ and $ys \preceq xs$?

Now we have a way to define them properly:

$$(\sqsubseteq) : A^* \leftarrow A^*$$
$$(\sqsubseteq) = (\![nil, cons \cup nil]\!)$$

and

$$(\preceq) : A^* \leftarrow A^*$$
$$(\preceq) = (\![nil, cons \cup \pi_2]\!)$$

where $\begin{cases} nil\ \_ = [\,] \\ cons\ (h, t) = h : t \end{cases}$ make up the **initial algebra** of finite lists:

$$\mathbf{in} = [nil, cons]$$

# Inductive relations thanks to $\mathbb{J} \dashv \mathbb{P}$

Recalling *take*, now we see where (6) came from:

$$(\sqsubseteq) = (\![nil, cons \cup nil]\!)$$

$$\Leftrightarrow \quad \{ \text{ universal property above } \}$$

$$\left\{ \begin{array}{l} (\sqsubseteq) \cdot nil = nil \\ (\sqsubseteq) \cdot cons = (cons \cup nil) \cdot (id \times (\sqsubseteq)) \end{array} \right.$$

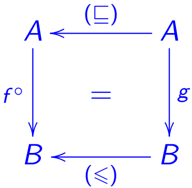
$$\Leftrightarrow \quad \{ \text{ go pointwise } \}$$

$$\left\{ \begin{array}{l} y \sqsubseteq [\,] \Leftrightarrow y = [\,] \\ y \sqsubseteq (h : t) \Leftrightarrow y = [\,] \vee \langle \exists\, t' \ : \ y = h : t' : \ t' \sqsubseteq t \rangle \end{array} \right.$$

# Back to Galois connections — in $\mathfrak{R}$

Remember GC $\quad f\ b \sqsubseteq a \Leftrightarrow b \leqslant g\ a$?

Now, every component of the GC — $f$, $g$, $(\sqsubseteq)$ and $(\leqslant)$ — is a **morphism** in $\mathfrak{R}$ and:




(42)

$$f \dashv g \quad \Leftrightarrow \quad f^\circ \cdot (\sqsubseteq) = (\leqslant) \cdot g$$

**NB**: $R^\circ$ is the converse of $R$, which always exists in $\mathfrak{R}$ — but not in the original $\mathfrak{S}$.

# More about this

See e.g. my talk

> *On the power of adjoint recursion. Contributed talk to IFIP WG 2.1 Short On-line Meeting #O6, 26 October 2021.*

Several more examples also in

> *Ralf Hinze. Adjoint folds and unfolds — an extended study. Science of Computer Programming, 78(11): 2108–2159, 2013.*

which inspired this work.

# Wrapping up

Original motivation was Ralf Hinze (2013):

---

*(...) Finally, we have left the exploration of* **relational** *adjoint (un)folds to future work.*

---

As shown, doing this leads to the algebra of inductive relations.

Altogether,

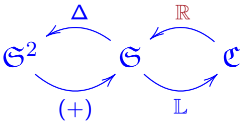
- I have learned to appreciate "adjoint folds" even more.
- **Adjunctions** are a very fertile device for structuring the MPC — **teaching** them (inc. **Galois connections**) should be mainstream.
- Current work: "adjoint folds" in language semantics and in linear algebra.

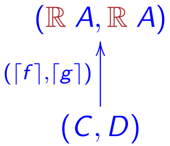
# Final quote

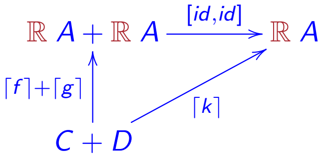
"My experience has been that theories are often more structured and more interesting when they are based on the real problems; somehow they are more exciting than completely abstract theories will ever be." *Donald Knuth (1973)*

# Appendix

# Composing $(+) \dashv \Delta$ and $\mathbb{L} \dashv \mathbb{R}$




$$\left\{ \begin{array}{l} \lceil f \rceil = \lceil k \rceil \cdot i_1 \\ \lceil g \rceil = \lceil k \rceil \cdot i_2 \end{array} \right. \quad \Leftrightarrow \quad \lceil k \rceil = [\lceil f \rceil, \lceil g \rceil]$$

# “Chemistry” between $\mathbb{L} \dashv \mathbb{R}$ and coproducts

$$\lceil k \rceil = [\lceil f \rceil, \lceil g \rceil]$$

$\Leftrightarrow$ { universal property }

$$\begin{cases} \lceil k \rceil \cdot i_1 = \lceil f \rceil \\ \lceil k \rceil \cdot i_2 = \lceil g \rceil \end{cases}$$

$\Leftrightarrow$ { fusion (15) twice }

$$\begin{cases} k \cdot \mathbb{L}\ i_1 = f \\ k \cdot \mathbb{L}\ i_2 = g \end{cases}$$

$\Leftrightarrow$ { coproducts }

$$k \cdot \underbrace{[\mathbb{L}\ i_1, \mathbb{L}\ i_2]}_{\delta} = [f, g]$$

$\Leftrightarrow$ { isomorphism $\delta$ }

$$k = [f, g] \cdot \delta^\circ$$

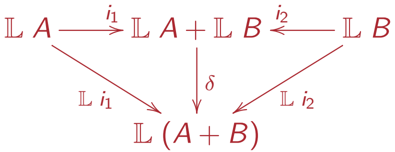
How can we be sure $\delta$ is an isomorphism?

# Limits and colimits

Left adjoints $\mathbb{L}$ preserve colimits, and thus **coproducts**:

$$\mathbb{L}\ (A+B) \underset{\delta}{\overset{\delta^\circ}{\rightleftarrows}} \mathbb{L}\ A + \mathbb{L}\ B \quad (\cong) \qquad \delta = [\mathbb{L}\ i_1, \mathbb{L}\ i_2]$$

Diagram:




Example: $(\mathbb{L}\ X = X \times K)$

$$(A+B) \times K \underset{\delta = undistl}{\overset{\delta^\circ = distl}{\rightleftarrows}} A \times K + B \times K \quad (\cong)$$

# Limits and colimits

Left adjoints $\mathbb{L}$ preserve colimits, and thus **coproducts**:

$$\mathbb{L}\,(A+B) \underset{\delta}{\overset{\delta^\circ}{\rightleftarrows}} \mathbb{L}\,A + \mathbb{L}\,B \qquad (\cong) \qquad \delta = [\mathbb{L}\,i_1, \mathbb{L}\,i_2]$$

Diagram:

$$\begin{array}{ccccc} \mathbb{L}\,A & \xrightarrow{i_1} & \mathbb{L}\,A + \mathbb{L}\,B & \xleftarrow{i_2} & \mathbb{L}\,B \\ & {\scriptstyle \mathbb{L}\,i_1} \searrow & \downarrow {\scriptstyle \delta} & \swarrow {\scriptstyle \mathbb{L}\,i_2} & \\ & & \mathbb{L}\,(A+B) & & \end{array}$$

Example: $(\mathbb{L}\,X = X \times K)$

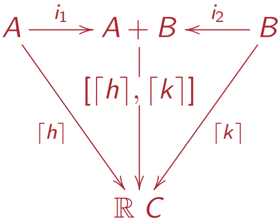
$$(A+B)\times K \underset{\delta = undistl}{\overset{\delta^\circ = distl}{\rightleftarrows}} A\times K + B \times K \qquad (\cong)$$

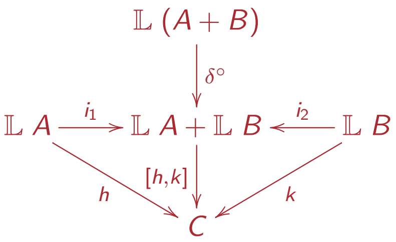
# "Chemistry" between $\mathbb{L} \dashv \mathbb{R}$ and coproducts

In summary:

$$[\lceil h \rceil, \lceil k \rceil] = \lceil [h, k] \cdot \delta^\circ \rceil \qquad (43)$$

Diagrams:



# Examples

For $\mathbb{L} \dashv \mathbb{R} := (\times K) \dashv (\_^K)$

(covariant exponentials), $[\lceil h \rceil, \lceil k \rceil] = \lceil [h,k] \cdot \delta^\circ \rceil$ (43) becomes

$$[\mathbf{curry}\, f, \mathbf{curry}\, g] = \mathbf{curry}\, ([f,g] \cdot distl) \quad (44)$$

For $\mathbb{L} \dashv \mathbb{R} := \mathbb{J} \dashv \mathbb{P}$

$\delta$ is the identity (relation) and so (43) becomes:

$$\Lambda[R,S] = [\Lambda R, \Lambda S] \quad (45)$$

Thus **relational coproducts** can be defined by:

$$[R,S] = \in \cdot [\Lambda R, \Lambda S]$$

# Examples

---

For $\mathbb{L} \dashv \mathbb{R} := (\times K) \dashv (\_^K)$

(covariant exponentials), $[\lceil h \rceil, \lceil k \rceil] = \lceil [h, k] \cdot \delta^\circ \rceil$ (43) becomes

$$[\mathbf{curry}\, f\,, \mathbf{curry}\, g] = \mathbf{curry}\,([f\,, g] \cdot distl) \tag{44}$$

---

For $\mathbb{L} \dashv \mathbb{R} := \mathbb{J} \dashv \mathbb{P}$

$\delta$ is the identity (relation) and so (43) becomes:

$$\Lambda[R, S] = [\Lambda R, \Lambda S] \tag{45}$$

Thus **relational coproducts** can be defined by:

$$[R, S] = \in \cdot [\Lambda R, \Lambda S]$$

# Dual theorem

$$\mathbb{R}\ out \cdot f = \phi \cdot \mathbb{G}\ f \cdot h \quad \Leftrightarrow \quad \lfloor f \rfloor = [\!(\lfloor \phi \cdot \mathbb{G}\ \eta \cdot h \rfloor)\!] \tag{46}$$

Calculation:

$$\lfloor f \rfloor = [\!(\lfloor \phi \cdot \mathbb{G}\ \eta \cdot h \rfloor)\!]$$

$$\Leftrightarrow \qquad \{ \text{ ana-universal } \}$$

$$out \cdot \lfloor f \rfloor = \mathbb{F}\ \lfloor f \rfloor \cdot \lfloor \phi \cdot \mathbb{G}\ \eta \cdot h \rfloor$$

$$\Leftrightarrow \qquad \{ \text{ fusion (23) twice } \}$$

$$\lfloor \mathbb{R}\ out \cdot f \rfloor = \lfloor \mathbb{R}\ \mathbb{F}\ \lfloor f \rfloor \cdot \phi \cdot \mathbb{G}\ \eta \cdot h \rfloor$$

$$\Leftrightarrow \qquad \{ \text{ isomorphism } \lfloor \_ \rfloor \text{ ; natural-}\phi \text{ } \}$$

$$\mathbb{R}\ out \cdot f = \phi \cdot \mathbb{G}\ \mathbb{R}\ \lfloor f \rfloor \cdot \mathbb{G}\ \eta \cdot h$$

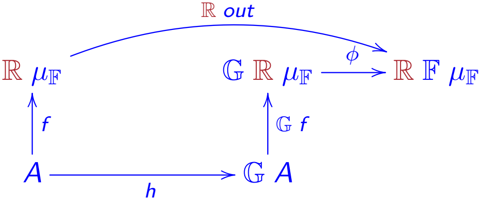
$$\Leftrightarrow \qquad \{ \text{ functor } \mathbb{G}\text{; cancellation } \mathbb{R}\ \lfloor f \rfloor \cdot \eta = f \text{ (22) } \}$$

$$\mathbb{R}\ out \cdot f = \phi \cdot \mathbb{G}\ f \cdot h$$

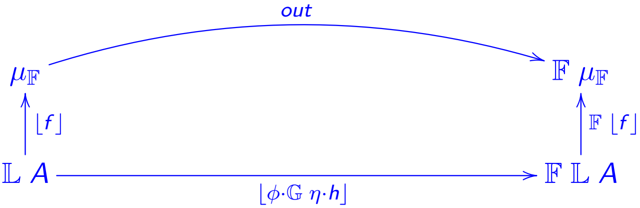
$\square$

# Dual theorem — diagram

$\mathbb{G}$-hylomorphism




$\mathbb{F}$-anamorphism :




$$\lfloor f \rfloor = [\![ \lfloor \phi \cdot \mathbb{G}\ \eta \cdot h \rfloor ]\!]$$

# Monads

A **monad**

$$A \xrightarrow{\eta} \mathbb{M} A \xleftarrow{\mu} \mathbb{M}^2 A$$

arises from any adjunction, where:

$$\mathbb{M} = \mathbb{R} \cdot \mathbb{L}$$
$$\eta = \lceil id \rceil$$
$$\mu = \mathbb{R}\ \epsilon$$

Monadic laws come straight from the adjunction laws.

Unit:

$$\mu \cdot \eta = id = \mu \cdot \mathbb{M}\ \eta$$

$\Leftrightarrow$ { $\mu = \mathbb{R}\ \epsilon$, $\eta = \lceil id \rceil$ etc }

$$\mathbb{R}\ \epsilon \cdot \lceil id \rceil = id = \mathbb{R}\ \epsilon \cdot (\mathbb{R}\ \mathbb{L}\ \eta)$$

$\Leftrightarrow$ { absorption (16); functor $\mathbb{R}$ }

$$\lceil \epsilon \rceil = id = \mathbb{R}\ (\epsilon \cdot \mathbb{L}\ \eta)$$

$\Leftrightarrow$ { reflection (12); cancellation (28) }

*true*

□

# Monads

A **monad**

$$A \xrightarrow{\eta} \mathbb{M} A \xleftarrow{\mu} \mathbb{M}^2 A$$

arises from any adjunction, where:

$$\mathbb{M} = \mathbb{R} \cdot \mathbb{L}$$
$$\eta = \lceil id \rceil$$
$$\mu = \mathbb{R}\ \epsilon$$

Monadic laws come straight from the adjunction laws.

Unit:

$$\mu \cdot \eta = id = \mu \cdot \mathbb{M}\ \eta$$

$\Leftrightarrow$ { $\mu = \mathbb{R}\ \epsilon$, $\eta = \lceil id \rceil$ etc }

$$\mathbb{R}\ \epsilon \cdot \lceil id \rceil = id = \mathbb{R}\ \epsilon \cdot (\mathbb{R}\ \mathbb{L}\ \eta)$$

$\Leftrightarrow$ { absorption (16); functor $\mathbb{R}$ }

$$\lceil \epsilon \rceil = id = \mathbb{R}\ (\epsilon \cdot \mathbb{L}\ \eta)$$

$\Leftrightarrow$ { reflection (12); cancellation (28) }

*true*

□

# Monad

Multiplication:

$$\mu \cdot \mu = \mu \cdot \mathbb{M}\ \mu$$

$$\Leftrightarrow \quad \{\ \mu = \mathbb{R}\ \epsilon;\ \text{functor}\ \mathbb{R}\ \}$$

$$\mathbb{R}\ (\epsilon \cdot \epsilon) = (\mathbb{R}\ \epsilon) \cdot (\mathbb{R}\ (\mathbb{L}\ (\mathbb{R}\ \epsilon)))$$

$$\Leftrightarrow \quad \{\ \text{functor}\ \mathbb{R}\ \}$$

$$\mathbb{R}\ (\epsilon \cdot \epsilon) = \mathbb{R}\ (\epsilon \cdot \mathbb{L}\ (\mathbb{R}\ \epsilon))$$

$$\Leftrightarrow \quad \{\ \text{natural-}\epsilon\ (17)\ \}$$

$$\mathbb{R}\ (\epsilon \cdot \epsilon) = \mathbb{R}\ (\epsilon \cdot \epsilon)$$

$\square$

# Kleisli composition

From the usual definition of **Kleisli composition**,

$f \bullet g = \mu \cdot \mathbb{M}\ f \cdot g$

(aside) we can infer:

$f \bullet g = \lceil \lfloor f \rfloor \cdot \lfloor g \rfloor \rceil$

$$
\begin{aligned}
& f \bullet g \\
= & \quad \{\ f \bullet g = \mu \cdot \mathbb{M}\ f \cdot g\ \} \\
& \mu \cdot \mathbb{M}\ f \cdot g \\
= & \quad \{\ \mathbb{M} = \mathbb{R} \cdot \mathbb{L};\ \mu = \mathbb{R}\ \epsilon\ \} \\
& \mathbb{R}\ \epsilon \cdot (\mathbb{R}\ (\mathbb{L}\ f)) \cdot g \\
= & \quad \{\ \text{functor } \mathbb{R}\ \} \\
& \mathbb{R}\ (\epsilon \cdot \mathbb{L}\ f) \cdot g \\
= & \quad \{\ \text{cancellation: } \epsilon \cdot \mathbb{L}\ f = \lfloor f \rfloor;\ g = \lceil \lfloor g \rfloor \rceil\ \} \\
& \mathbb{R}\ \lfloor f \rfloor \cdot \lceil \lfloor g \rfloor \rceil \\
= & \quad \{\ \text{absorption: } (\mathbb{R}\ g) \cdot \lceil h \rceil = \lceil g \cdot h \rceil\ \} \\
& \lceil \lfloor f \rfloor \cdot \lfloor g \rfloor \rceil
\end{aligned}
$$

# Other relational hylos and their adjoints

Example: **list membership**

$$\begin{cases} a \ \epsilon \ [\,] = \mathit{False} \\ a \ \epsilon \ (h : t) = (a = h) \vee a \ \epsilon \ t \end{cases}$$

is the relational **hylo**

$$\epsilon = [\bot, \pi_1 \cup \epsilon \cdot \pi_2] \cdot \mathbf{in}^\circ \tag{47}$$

**NB**: not the relational **cata** $\epsilon = (|[\bot, \pi_1 \cup \pi_2]|)$ that one might feel tempted to write... which is the empty relation!

# Other relational hylos and their adjoints

ot a cata... But perhaps this hylo (47) has an **adjoint** cata? Yes, since

$$\epsilon = [\bot, \pi_1 \cup \epsilon \cdot \pi_2] \cdot \mathbf{in}^\circ$$

unfolds into

$$\epsilon \cdot \mathbf{in} = \underbrace{[\bot, [id, id]]}_{R} \cdot \underbrace{id + (id + \epsilon)}_{\mathbb{G}\ \epsilon} \cdot \underbrace{id + (i_1 \cdot \pi_1 \cup i_2 \cdot \pi_2)}_{\Phi}$$

where the core of

$$\Phi : \underbrace{1 + A \times A^*}_{\mathbb{F}\ A^*} \to \underbrace{1 + (A + A^*)}_{\mathbb{G}\ A^*}$$

is the (disjoint) union of the two projections $\pi_1 \cup \pi_2$.

# Relational hylos and their adjoints

What is its adjoint? Not surprisingly:

$$\Lambda\epsilon = (\!| \Lambda[\bot, \pi_1 \cup \in \cdot\, \pi_2] |\!)$$

$$\Leftrightarrow \qquad \{ \ \mathbb{P}\text{-transpose of coproducts (45)} \ \}$$

$$\Lambda\epsilon = (\!| [\Lambda\bot, \Lambda(\pi_1 \cup \in \cdot\, \pi_2)] |\!)$$

$$\Leftrightarrow \qquad \{ \ \text{introduce } \mathit{join} \text{ etc (see below)} \ \}$$

$$\Lambda\epsilon = (\!| [\underline{\{\,\}}, \mathit{join}] |\!)$$

$$\Leftrightarrow \qquad \{ \ \text{introduce } \mathit{elems} \ \}$$

$$\Lambda\epsilon = \mathit{elems} \tag{48}$$

where

$$\begin{cases} \mathit{elems}\ [\,] = \{\,\} \\ \mathit{elems}\ (h : t) = \{\, h \,\} \cup \mathit{elems}\ t \end{cases}$$

# Relational hylos and their adjoints

Details:

$$elems = (\![\underline{\{\,\}}, join]\!) \quad \Leftrightarrow \quad \left\{ \begin{array}{l} elems\ [\,] = \{\,\} \\ elems\ (h : t) = \{\, h\,\} \cup elems\ t \end{array} \right.$$

where

$$join\ (a, s) = \{\, a\,\} \cup s$$

since:

$$join = \Lambda(\pi_1 \cup \in \cdot\, \pi_2)$$
$$\Lambda(R \cup S)\ a = (\Lambda R\ a) \cup (\Lambda S\ a)$$
$$\Lambda{\in} = id$$

etc.

Usual way of doing list membership: $\epsilon = \in \cdot\ elems$, cf. (48).

# (Contravariant) exponentials: $(K-) \dashv (K-)$

Isomorphism

$$\mathbb{L}\,A \to B \;\underset{\lfloor\_\rfloor}{\overset{\lceil\_\rceil}{\cong}}\; A \to \mathbb{R}\,B$$

becomes (note the arrows reversed on the left side)

$$K^A \leftarrow B \;\underset{\mathit{flip}}{\overset{\mathit{flip}}{\cong}}\; A \to K^B$$

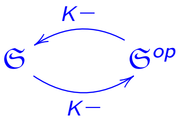
recalling (Haskell):

```
flip :: (a -> b -> c) -> b -> a -> c
flip f b a = f a b
```

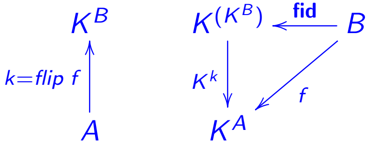
# (Contravariant) exponentials: $(K-) \dashv (K-)$

Contravariant **self-adjunction**. More formally:

$$\begin{cases} \mathbb{L}\ X = K^X \\ \mathbb{R}\ X = K^X \\ \epsilon = \mathbf{fid} = \mathit{flip}\ id \end{cases} \qquad \begin{cases} \lceil f \rceil = \mathit{flip}\ f \\ \lfloor f \rfloor = \mathit{flip}\ f \end{cases}$$




$$k = \mathit{flip}\ f \quad \Leftrightarrow \quad f = \underbrace{K^k \cdot \mathbf{fid}}_{\mathit{flip}\ k}$$



# (Contravariant) exponentials: $(K-) \dashv (K-)$

Contravariant **exponential functor**:

$$\mathfrak{S} \xrightarrow{K-} \mathfrak{S}^{op}$$

$$\begin{cases} K^{(-)} : (A \to B) \to (B \to K) \to A \to K \\ K^k \, g = g \cdot k \end{cases}$$

$$A \xrightarrow{k} B \qquad K^B \xrightarrow{K^k} K^A$$

That is:

$$K^k = (\cdot k) \tag{49}$$

# References

R. Bird and O. de Moor. *Algebra of Programming*. Series in Computer Science. Prentice-Hall, 1997.

Ralf Hinze. Adjoint folds and unfolds — an extended study. *Science of Computer Programming*, 78(11):2108–2159, 2013. ISSN 0167-6423.

D.E. Knuth. The dangers of computer-science theory. *Studies in Logic and the Foundations of Mathematics*, 74:189–195, 1973.

J.N. Oliveira. A note on the under-appreciated for-loop. Technical Report TR-HASLab:01:2020 (PDF), HASLab/U.Minho and INESC TEC, 2020.