2IPH0 – Lecture 9 – Monads

Functional Programming, 2021–2022, Q1

Tom Verhoeff

Lecture overview

Recap

Covered so far

To be covered

Covered so far

- Function and type combinators: $\circ$, $\vartriangle$, $\times$, $\triangledown$, $+$
- Polymorphic type judgment
- Pointwise and pointfree characterization of $\vartriangle$ and $\triangledown$
- Techniques for recursive function definitions (accumulation, tupling)
- Catamorphisms on $\mathbb{N}$ and $\mathbb{L}$, and their pointwise/pointfree char'n
  Paramorphisms, logarithmic fold on $\mathbb{N}$
- Cata-fusion theorems for catas on $\mathbb{N}$ and $\mathbb{L}$
- Corresponding calculation techniques

To be covered

- Monads (intro; only tested in Assignment 2; not in final test)
- General theory of inductive types
  Polynomial/Kleene functors, F-algebras, F-homomorphisms
- General theory of co-inductive types
- Hylomorphisms
- Streams (infinite co-data)
- Lambda Calculus (not tested)

Intermezzo on Function Composition

Various ways of combining ('gluing') functions

Functions that don't combine

Various ways of combining ('gluing') functions

$f \in$	$g \in$	combinator	composite $\in$	name
$C \leftarrow B$	$B \leftarrow A$	$f \circ g$	$C \leftarrow A$	composition
$B \leftarrow A$	$C \leftarrow A$	$f \vartriangle g$	$B \times C \leftarrow A$	split
$C \leftarrow A$	$D \leftarrow B$	$f \times g$	$C \times D \leftarrow A \times B$	product
$C \leftarrow A$	$C \leftarrow B$	$f \triangledown g$	$C \leftarrow A + B$	case
$C \leftarrow A$	$D \leftarrow B$	$f + g$	$C + D \leftarrow A + B$	sum

Currying and uncurrying can also help to make functions composable

Functions that don't combine

- Starting point: functions of type $X \leftarrow X$, that can be composed easily
- Slight variants break composability:
  - Partial function signals failure: $\mathbb{1} + X \leftarrow X$ (return $\hookrightarrow.\_$ for failure)
  - Partial function throws exception: $E + X \leftarrow X$ (return $\hookrightarrow.e$ for exception e)
  - Function produces no or multiple results: $\mathbb{L}.X \leftarrow X$ (return list of results)
  - Function logs tracing info: $X \times T \leftarrow X$ (return tuple of result and info)
- One approach: 'Extend' domain to match codomain, and fix it on the inside
  - 'Extend' *each function definition* to handle 'extended' arguments (not DRY)
- Two other approaches: fix it on the outside
  1. Define custom composition combinator, a.k.a. Kleisli composition
  2. Introduce extension decorator (with function-to-be-extended as parameter)

Functions $\mathbb{1} + X \leftarrow X$ (with failure notification)

TU/e

Fix on the inside, for $\mathbb{1} + X \leftarrow X$ (failure notification)

- Given $f \in \mathbb{1} + X \leftarrow X$ and $g \in \mathbb{1} + X \leftarrow X$, we want $f \circ g$ 'as expected'
- To allow $f' \circ g$, define 'extended' $f' \in \mathbb{1} + X \leftarrow \mathbb{1} + X$ by

$$\begin{array}{lcll} f'.(\hookrightarrow.\_) & = & \hookrightarrow.\_ & \{\text{ failure persists }\} \\ f'.(\hookleftarrow.x) & = & f.x & \{\text{ normal flow (can also lead to failure) }\} \end{array}$$

- Given $f \in \mathbb{1} + X \leftarrow X \times X$ and $g, h \in \mathbb{1} + X \leftarrow X$, we want $f \circ (g \vartriangle h)$
- To allow $f' \circ (g \vartriangle h)$, define 'extended' $f' \in \mathbb{1} + X \leftarrow (\mathbb{1} + X) \times (\mathbb{1} + X)$ by

$$\begin{array}{lcll} f'.(\hookrightarrow.\_,\ \ldots) & = & \hookrightarrow.\_ & \{\text{ left failure persists }\} \\ f'.(\ldots,\ \hookrightarrow.\_) & = & \hookrightarrow.\_ & \{\text{ right failure persists }\} \\ f'.(\hookleftarrow.x,\ \hookleftarrow.y) & = & f.(x, y) & \{\text{ normal flow }\} \end{array}$$

- What a drag (to do this for every function)

TU/e

Fix on the outside (1), for $\mathbb{1} + X \leftarrow X$: Kleisli composition

- Define Kleisli composition $\bullet$ for $f, g \in \mathbb{1} + X \leftarrow X$ by

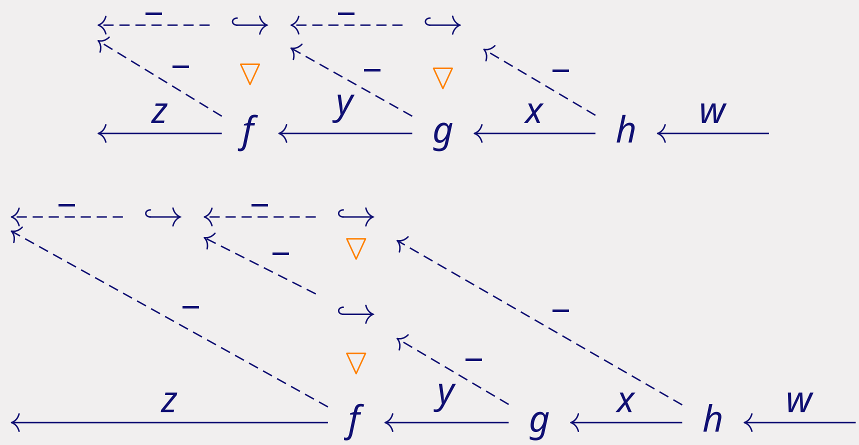
$$
\begin{array}{rcll}
f \bullet g & \in & \mathbb{1} + X \leftarrow X & \\
f \bullet g & = & (\hookrightarrow \triangledown f) \circ g & \{ \text{pointfree} \} \\
(f \bullet g).x & = & f'.(g.x) \ \textbf{where} & \{ \text{pointwise} \} \\
f'.(\hookrightarrow.\_) & = & \hookrightarrow.\_ & \\
f'.(\hookleftarrow.y) & = & f.y &
\end{array}
$$

- This captures that a failure from g bypasses f
- In functional programming, you can define your own exception handling

Kleisli composition for $\mathbb{1} + X \leftarrow X$ is associative

$$
\begin{array}{ll}
& f \bullet (g \bullet h) \qquad \{ \text{goal: } (f \bullet g) \bullet h \} \\
= & \{ \text{def. } \bullet \text{ (twice)} \} \\
& (\hookrightarrow \triangledown f) \circ (\hookrightarrow \triangledown g) \circ h \\
= & \{ \ \triangledown\text{-fusion: } p \circ (q \triangledown r) = (p \circ q) \triangledown (p \circ r) \} \\
& (\ (\hookrightarrow \triangledown f) \circ \hookrightarrow \ \triangledown \ (\hookrightarrow \triangledown f) \circ g \ ) \circ h \\
= & \{ \ \triangledown\text{-self: } (p \triangledown q) \circ \hookrightarrow = p \} \\
& (\hookrightarrow \ \triangledown \ (\hookrightarrow \triangledown f) \circ g \ ) \circ h \\
= & \{ \ \text{def. } \bullet \} \\
& (\ \hookrightarrow \ \triangledown \ f \bullet g \ ) \circ h \\
= & \{ \ \text{def. } \bullet \} \\
& (f \bullet g) \bullet h
\end{array}
$$

Associativity by circuits



Results on ‘cut’ into f differ: $\mathbb{1}+X$ vs $\mathbb{1}+(\mathbb{1}+X)$

Kleisli composition for $\mathbb{1}+X \leftarrow X$ has unit element

- Define u by

$$u \in \mathbb{1}+X \leftarrow X$$
$$u = \hookleftarrow \qquad \{\text{ pass value on as success, never failing }\}$$

- Verify by (parallel/tupled) calculation:

$$\begin{aligned}
& (\, f \bullet u,\ u \bullet f \,) \qquad \{\text{ goal: } (f, f) \}\\
= \ & \{\text{ def. } \bullet,\ u \}\\
& (\ (\hookrightarrow \triangledown f) \circ \hookleftarrow,\ (\hookrightarrow \triangledown \hookleftarrow) \circ f\ )\\
= \ & \{\ \triangledown\text{-self},\ \triangledown\text{-id}\ \}\\
& (\, f,\ \mathrm{id} \circ f \,)\\
= \ & \{\text{ id is unit of composition }\}\\
& (\, f, f \,)
\end{aligned}$$

Generalize to $\mathbb{1} + Y \leftarrow X$: Maybe monad $\mathbb{M}.X = \mathbb{1} + X$

- Decouple domain and codomain (or: make coupling explicit and general)
- Define (polymorphic) type $\mathbb{M}$ by $\mathbb{M}.X = \mathbb{1} + X$, used for codomains
- $f \in \mathbb{M}.C \leftarrow B$ and $g \in \mathbb{M}.B \leftarrow A$
- Kleisli composition $f \bullet g \in \mathbb{M}.C \leftarrow A$ is defined by $f \bullet g = (\hookrightarrow \triangledown f) \circ g$
- N.B. Here: $\hookrightarrow \in \mathbb{M}.C \leftarrow \mathbb{1}$; hence $(\hookrightarrow \triangledown f) \in \mathbb{M}.C \leftarrow \mathbb{M}.B$
- Unit of Kleisli composition: $u \in \mathbb{M}.X \leftarrow X$ is defined by $u = \hookleftarrow$
- $(\mathbb{M}, \bullet, u)$ is (one manifestation of) the Maybe monad
- Maybe monad offers regular composition *plus side channel* for failures
- $\bullet \in (\mathbb{M}.C \leftarrow A) \leftarrow (\mathbb{M}.C \leftarrow B) \times (\mathbb{M}.B \leftarrow A)$ { Haskel notation: <=< }
- Kleisli composition ensures that failures combine properly (i.e., persist)

Functions $\mathbb{L}.X \leftarrow X$ (with result lists)

Fix on the inside

Fix on the outside (1): Kleisli composition

Kleisli composition is associative

Kleisli composition has unit element

Generalize to $\mathbb{L}.Y \leftarrow X$: List monad $\mathbb{L}$

Fix on the inside, for $\mathbb{L}.X \leftarrow X$ (result lists)

- Given $f \in \mathbb{L}.X \leftarrow X$ and $g \in \mathbb{L}.X \leftarrow X$, we want $f \circ g$ 'as expected'
  $\mathbb{L}.X$ generalizes $\mathbb{1} + X$: $\hookrightarrow.\_ \sim []$ (failure) and $\hookleftarrow.x \sim [x]$ (success)
- To allow $f' \circ g$, define 'extended' $f' \in \mathbb{L}.X \leftarrow \mathbb{L}.X$ by

$$\begin{aligned} f' &= concat \circ map.f && \{\text{ a.k.a. } flatmap.f \ \} \\ f'.xs &= [y \mid x \leftarrow xs,\ y \leftarrow f.x] && \{\text{ pointwise with list comprehension }\} \end{aligned}$$

  Try in Haskell: `f x = [x, x]; fe = concat . map f; fe [3, 4]`
- Given $f \in \mathbb{L}.X \leftarrow X \times X$ and $g, h \in \mathbb{L}.X \leftarrow X$, we want $f \circ (g \vartriangle h)$
- To allow $f' \circ (g \vartriangle h)$, define 'extended' $f' \in \mathbb{L}.X \leftarrow \mathbb{L}.X \times \mathbb{L}.X$ by

$$f'.(xs, ys) = [z \mid x \leftarrow xs,\ y \leftarrow ys,\ z \leftarrow f.(x, y)]$$

- What a drag (to do this for every function)

Fix on the outside (1), for $\mathbb{L}.X \leftarrow X$: Kleisli composition

- Define Kleisli composition $\bullet$ for $f, g \in \mathbb{L}.X \leftarrow X$ by

$$\begin{aligned} f \bullet g &\in \mathbb{L}.X \leftarrow X \\ f \bullet g &= concat \circ map.f \circ g && \{\text{ pointfree }\} \\ (f \bullet g).x &= concat.(map.f.(g.x)) && \{\text{ pointwise }\} \\ &= [z \mid y \leftarrow g.x,\ z \leftarrow f.y] \end{aligned}$$

- This captures that f is applied to each result from g, returned in one list

Kleisli composition for $\mathbb{L}.X \leftarrow X$ is associative

$$
\begin{array}{ll}
 & f \bullet (g \bullet h) \qquad \{\text{goal: } (f \bullet g) \bullet h\,\} \\
= & \{\ \text{def. } \bullet \text{ (rightmost)}\,\} \\
 & f \bullet (concat \circ map.g \circ h) \\
= & \{\ \text{def. } \bullet\,\} \\
 & concat \circ map.f \circ concat \circ map.g \circ h \\
= & \{\ \text{fusion (exercise)}\,\} \\
 & concat \circ map.\,(concat \circ map.f \circ g) \circ h \\
= & \{\ \text{def. } \bullet\,\} \\
 & concat \circ map.\,(f \bullet g) \circ h \\
= & \{\ \text{def. } \bullet\,\} \\
 & (f \bullet g) \bullet h
\end{array}
$$

TU/e

Kleisli composition for $\mathbb{L}.X \leftarrow X$ has unit element

- Define u by

$$
\begin{array}{lll}
u & \in & \mathbb{L}.X \leftarrow X \\
u & = & (\vdash[\,]) \qquad \{\,\text{singleton, for single result: } u.x = [\,x\,]\,\}
\end{array}
$$

- Verify by (parallel/tupled) calculation:

$$
\begin{array}{ll}
 & (\,f \bullet u,\ u \bullet f\,) \qquad \{\,\text{goal: } (f,f)\,\} \\
= & \{\,\text{def. } \bullet,\ u\,\} \\
 & (concat \circ map.f \circ (\vdash[\,]),\ concat \circ map.(\vdash[\,]) \circ f\,) \\
= & \{\,\text{pointwise (left), fusion (right, exercise)}\,\} \\
 & (\,f,\ \mathrm{id} \circ f\,) \\
= & \{\,\text{id is unit of composition}\,\} \\
 & (\,f,\ f\,)
\end{array}
$$

TU/e

Generalize to $\mathbb{L}.Y \leftarrow X$: List monad $\mathbb{L}$

- Decouple domain and codomain
- Type $\mathbb{L}$ is polymorphic, used for codomains
- $f \in \mathbb{L}.C \leftarrow B$ and $g \in \mathbb{L}.B \leftarrow A$
- Kleisli composition $f \bullet g \in \mathbb{L}.C \leftarrow A$ is defined by $f \bullet g = concat \circ map.f \circ g$
- N.B. Here: $concat \in \mathbb{L}.C \leftarrow \mathbb{L}.(\mathbb{L}.C)$
- Unit of Kleisli composition: $u \in \mathbb{L}.X \leftarrow X$ is defined by $u = (\vdash[\,])$
- $(\mathbb{L}, \bullet, u)$ is (one manifestation of) the List monad
- List monad offers regular composition *plus side channel* for multi-results
- $\bullet \in (\mathbb{L}.C \leftarrow A) \leftarrow (\mathbb{L}.C \leftarrow B) \times (\mathbb{L}.B \leftarrow A)$ { Haskel notation: <=< }
- Kleisli composition ensures that result lists combine properly

Functions $X \times \mathbb{L}.T \leftarrow X$ (with logging)

Fix on the inside

Fix on the outside (1): Kleisli composition

Kleisli composition is associative

Associativity by circuits

Kleisli composition has unit element

Unit by circuits

Generalize to $Y \times \mathbb{L}.T \leftarrow X$: Writer monad $\mathbb{W}$

Fix on the inside, for $X \times \mathbb{L}.T \leftarrow X$ (with logging)

- Given $f \in X \times \mathbb{L}.T \leftarrow X$ and $g \in X \times \mathbb{L}.T \leftarrow X$, we want $f \circ g$ 'as expected'
- To allow $f' \circ g$, define 'extended' $f' \in X \times \mathbb{L}.T \leftarrow X \times \mathbb{L}.T$ by

$$\begin{array}{rcll} f' & = & (\mathrm{id} \times +\!\!+) \circ \mathit{assocr} \circ (f \times \mathrm{id}) \ \textbf{where} & \\ \mathit{assocr} & = & (\ll \circ \ll) \vartriangle (\gg \times \mathrm{id}) & \{ \mathit{assocr}((x,s),t) = (x,(s,t)) \} \\ f'.(x,t) & = & \textbf{let } (y,s) = f.x \textbf{ in } (y, s +\!\!+ t) & \{ \text{pointwise: prepend to log} \} \end{array}$$

- What a drag (to do this for every function)

TU/e

Fix on the outside (1), for $X \times \mathbb{L}.T \leftarrow X$: Kleisli composition

- Define Kleisli composition $\bullet$ for $f, g \in X \times \mathbb{L}.T \leftarrow X$ by

$$\begin{array}{rcll} f \bullet g & \in & X \times \mathbb{L}.T \leftarrow X & \\ f \bullet g & = & (\mathrm{id} \times +\!\!+) \circ \mathit{assocr} \circ (f \times \mathrm{id}) \circ g & \{ \text{pointfree} \} \\ (f \bullet g).x & = & \textbf{let } (y,t) = g.x & \\ & & \qquad (z,s) = f.y & \\ & & \textbf{in } (z, s +\!\!+ t) & \{ \text{pointwise} \} \end{array}$$

- This captures that f's logging is prepended to that from g

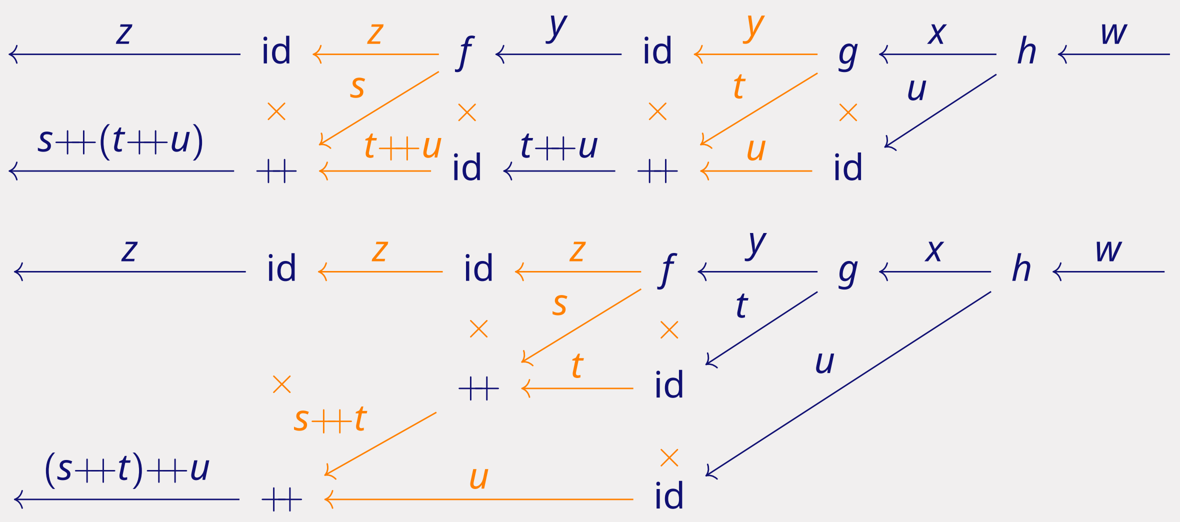
TU/e

Kleisli composition for $X \times \mathbb{L}.T \leftarrow X$ is associative

$$
\begin{array}{ll}
 & f \bullet (g \bullet h) \qquad \{\text{ goal: } (f \bullet g) \bullet h \,\} \\
= & \{\ \text{def.} \bullet \text{(rightmost)} \,\} \\
 & f \bullet (\ (\mathrm{id} \times +\!\!+) \circ \mathit{assocr} \circ (g \times \mathrm{id}) \circ h\ ) \\
= & \{\ \text{def.} \bullet\text{, composition is associative} \,\} \\
 & (\mathrm{id} \times +\!\!+) \circ \mathit{assocr} \circ (f \times \mathrm{id}) \circ (\mathrm{id} \times +\!\!+) \circ \mathit{assocr} \circ (g \times \mathrm{id}) \circ h \\
= & \{\ \text{fusion (exercise; easier than solving a Sudoku)} \,\} \\
 & (\mathrm{id} \times +\!\!+) \circ \mathit{assocr} \circ ((\ (\mathrm{id} \times +\!\!+) \circ \mathit{assocr} \circ (f \times \mathrm{id}) \circ g\ ) \times \mathrm{id}) \circ h \\
= & \{\ \text{def.} \bullet \,\} \\
 & (\mathrm{id} \times +\!\!+) \circ \mathit{assocr} \circ ((f \bullet g) \times \mathrm{id}) \circ h \\
= & \{\ \text{def.} \bullet \,\} \\
 & (f \bullet g) \bullet h
\end{array}
$$

TU/e

Associativity by circuits



Results on 'cut' into leftmost $\mathrm{id} \times +\!\!+$ differ: $(z,\ (s, t +\!\!+ u))$ vs $(z,\ (s +\!\!+ t, u))$

TU/e

Kleisli composition for $X \times \mathbb{L}.T \leftarrow X$ has unit element

- Define u by

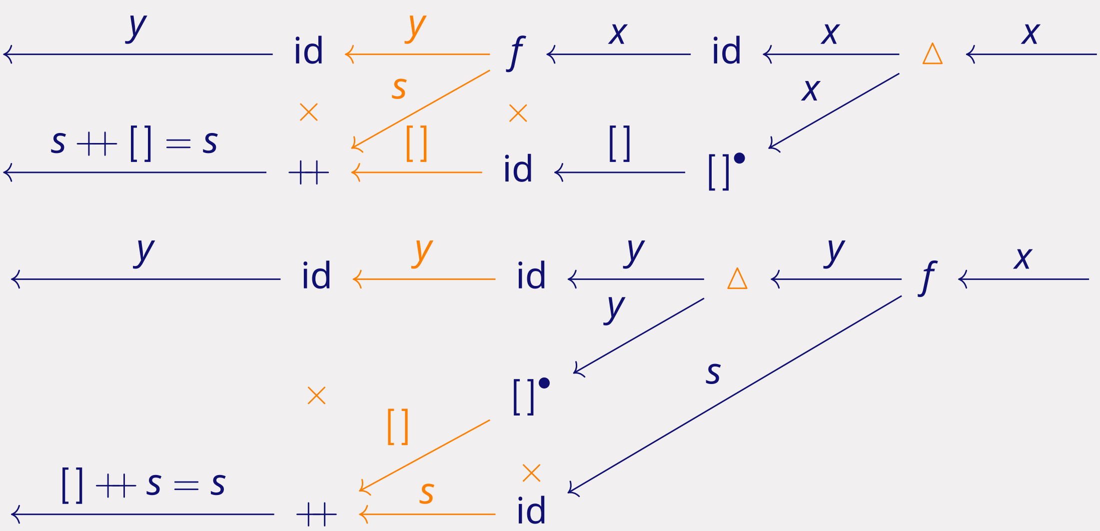
$$u \in X \times \mathbb{L}.T \leftarrow X$$

$$u = \mathrm{id} \vartriangle []^{\bullet} \qquad \{ \text{no logging data yet: } u.x = (x, []) \}$$

- Verify by (parallel/tupled) calculation:

$$
\begin{array}{ll}
 & (\, f \bullet u,\ u \bullet f \,) \qquad \{ \text{goal: } (f, f) \} \\
= & \{ \text{def. } \bullet,\ u \} \\
 & (\, (\mathrm{id} \times +\!\!+) \circ \mathit{assocr} \circ (f \times \mathrm{id}) \circ (\mathrm{id} \vartriangle []^{\bullet}), \\
 & \ \ (\mathrm{id} \times +\!\!+) \circ \mathit{assocr} \circ ((\mathrm{id} \vartriangle []^{\bullet}) \times \mathrm{id}) \circ f \,) \\
= & \{ \text{fusion, } [] \text{ (right/left) unit of } +\!\!+ \} \\
 & (\, f,\ \mathrm{id} \circ f \,) \\
= & \{ \mathrm{id} \text{ is unit of composition} \} \\
 & (\, f,\ f \,)
\end{array}
$$

Unit by circuits



Results on 'cut' from leftmost $\mathrm{id} \times +\!\!+$ differ: $(y,\ (s, []))$ vs $(y,\ ([], s))$

Generalize to $Y \times \mathbb{L}.T \leftarrow X$: Writer monad $\mathbb{W}$

- Decouple domain and codomain
- Define (polymorphic) type $\mathbb{W}$ by $\mathbb{W}.X = X \times \mathbb{L}.T$, used for codomains
- $f \in \mathbb{W}.C \leftarrow B$ and $g \in \mathbb{W}.B \leftarrow A$
- Kleisli composition $f \bullet g \in \mathbb{W}.C \leftarrow A$ is defined by

$$f \bullet g \quad = \quad (\mathrm{id} \times +\!\!+) \circ assocr \circ (f \times \mathrm{id}) \circ g$$

- Unit of Kleisli composition: $u \in \mathbb{W}.X \leftarrow X$ is defined by $u = \mathrm{id} \vartriangle [\,]^\bullet$
- $(\mathbb{W}, \bullet, u)$ is (one manifestation of) the Writer monad
- Writer monad offers regular composition *plus side channel* for logging
- $\bullet \in (\mathbb{W}.C \leftarrow A) \leftarrow (\mathbb{W}.C \leftarrow B) \times (\mathbb{W}.B \leftarrow A)$ { Haskel notation: <=< }
- Kleisli composition ensures that logs combine properly

Fix on the outside (2): bind

Fix on the outside (2): extend functions

Fix on the outside (2): extend functions

- Combinator (decorator) to ‘extend’ $f \in M.Y \leftarrow X$: take ‘extended’ argument
- Notation: $(\gg\!\!= f)$ (pronounce: ‘bind’ or ‘monadic apply’)
- $\gg\!\!= \;\in M.Y \leftarrow M.X \times (M.Y \leftarrow X)$ with $(\gg\!\!= f) \;=\; f \bullet \mathrm{id}$: $mx \gg\!\!= f \;=\; (f \bullet \mathrm{id}).mx$

Monad	$\mathbb{M}.X = \mathbb{1} + X$	$\mathbb{L}.X$	$\mathbb{W}.X = X \times \mathbb{L}.T$
$(\gg\!\!= f)$	$\hookrightarrow \triangledown f$	$\mathit{concat} \circ \mathit{map}.f$	$(\mathrm{id} \times +\!\!+) \circ \mathit{assocr} \circ (f \times \mathrm{id})$

- Can rewrite $\hookrightarrow \triangledown f = (\hookrightarrow \triangledown \mathrm{id}) \circ (\mathrm{id} + f)$ { $+$ -absorption }
- Note that

$$
\begin{aligned}
(\hookrightarrow \triangledown \mathrm{id}) \circ (\mathrm{id} + f) &= (\hookrightarrow \triangledown \mathrm{id}) \circ \mathbb{M}.f \\
\mathit{concat} \circ \mathit{map}.f &= \mathit{concat} \circ \mathbb{L}.f \\
(\mathrm{id} \times +\!\!+) \circ \mathit{assocr} \circ (f \times \mathrm{id}) &= (\mathrm{id} \times +\!\!+) \circ \mathit{assocr} \circ \mathbb{W}.f
\end{aligned}
$$

Monads dissected

A deeper look into monads

- Let $f \in M.C \leftarrow B$ and $g \in M.B \leftarrow A$
- Kleisli composition takes form $f \bullet g = \mu \circ M.f \circ g$
- Types: $M.C \xleftarrow{\mu} M.(M.C) \xleftarrow{M.f} M.B \xleftarrow{g} A$
- μ is flatten operator (a.k.a. *multiplication* or *join*): $\mu \in M.X \leftarrow M.(M.X)$
- Relationship to Kleisli composition: $\mu = \mathrm{id} \bullet \mathrm{id}$
- Monad can also be defined as triple (M, μ, u) *with some Monad laws*
- … or as triple $(M, \gg\!=, u)$ (in Haskell: u is named **return**) *with Monad laws*
  1. `(return x) >>= f ==== f x`
  2. `mx >>= return ==== mx`
  3. `(mx >>= f) >>= g ==== mx >>= (\x -> f x >>= g)`

Operator properties for monad M

$$
\begin{aligned}
(f \bullet g) \bullet h &= f \bullet (g \bullet h) \\
f \bullet u &= f \\
u \bullet f &= f \\
(f \bullet g) \circ h &= f \bullet (g \circ h) \\
(f \circ g) \bullet h &= f \bullet (M.g \circ h) \\
\mathrm{id} \bullet \mathrm{id} &= \mu \qquad \{\, \textit{join} \text{ or flatten} \,\} \\
f \bullet g &= \mu \circ M.f \circ g \qquad \{\, \text{Kleisli composition} \,\} \\
mx \gg\!= f &= (\mu \circ M.f).mx \qquad \{\, \text{bind, monadic application} \,\} \\
f \bullet g &= (\gg\!= f) \circ g \;=\; (\lambda x : g.x \gg\!= f) \\
mx \gg\!= f &= (f \bullet \mathrm{id}).mx \\
M.f &= (u \circ f) \bullet \mathrm{id} \qquad \{\, \text{a.k.a. } \textit{liftM.f} \,\} \\
\mu \circ u &= \mathrm{id} \qquad \{\, \text{mind the types: } M.X \leftarrow M.X \,\}
\end{aligned}
$$

Sequencing and do -notation

- Define:
  - $x \gg y \;=\; x \gg\!= y^{\bullet}$ { don't confuse this infix $\gg$ with projection }
  - $\mathbf{do}\,\{\, x \,\} \;=\; x$
  - $\mathbf{do}\,\{\, x; x_1; \ldots; x_n \,\} \;=\; x \gg \mathbf{do}\,\{\, x_1; \ldots; x_n \,\}$
  - $\mathbf{do}\,\{\, a \leftarrow x; x_1; \ldots; x_n \,\} \;=\; x \gg\!= (\lambda\, a : \mathbf{do}\,\{\, x_1; \ldots; x_n \,\})$ { x_i may contain a }
  - $[\, e \mid a_1 \leftarrow x_1, \ldots, a_n \leftarrow x_n \,] \;=\; \mathbf{do}\,\{\, a_1 \leftarrow x_1; \ldots; a_n \leftarrow x_n; u.e \,\}$ { list monad }

  where all expressions x and x_i map into the (same) monad
  If needed, the last one can use u to accomplish this
- Appropriate monad is used depending on types involved
- $(f \bullet g).x \;=\; \mathbf{do}\,\{\, y \leftarrow g.x; f.y \,\}$
- $\mathbf{do}\,\{\, a \leftarrow [1,2,3]; [\,a^2\,] \,\} \;=\; [\,a^2 \mid a \leftarrow [1,2,3]\,]$

Standard monads

- Identity: $M.X = X$ without side-effect
- Maybe: $M.X = \mathbb{1} + X$ with one failure signal
- Error: $M.X = E + X$ with multiple exceptions/errors
- List: $M.X = \mathbb{L}.X$ with multiple results (possibly none)
- Writer: $M.X = X \times T$ with logging
- IO: $M.X = \mathrm{IO}.X$ with input and output
- Reader: $M.X = E \to X$ with read-only environment
- State: $M.X = S \to X \times S$ with updatable state
- Continuation: $M.X = (A \to X) \to X$ with continuation chaining

Identity monad

- Identity monad defined by functor $F.X = X$ (identity functor)
- Kleisli composition: $f \bullet g = f \circ g$
- Unit of Kleisli composition: $u = \mathrm{id}$
- Identity monad offers regular composition *without side channel*

IO monad

- Input and output are/have side effects
- **IO** monad (in Haskell) encapsulates this
- `main :: IO ()`
- `putStrLn :: String -> IO ()`
- `getLine :: IO String`
- Get two line of input and print them in reverse order:

```
reverse2lines :: IO ()
reverse2lines = do line1 <- getLine
                   line2 <- getLine
                   putStrLn (reverse line2)
                   putStrLn (reverse line1)
```

Overview of generalization steps

- $X \leftarrow X$
- $\mathbb{1} + X \leftarrow X$ to signal failures
  If $f \in X \leftarrow X$, then $u \circ f \in \mathbb{1} + X \leftarrow X$ (never fails)
- $\mathbb{1} + X \leftarrow \mathbb{1} + X$ to allow composition
  If $f \in \mathbb{1} + X \leftarrow X$, then $(\gg\!= f) = f \bullet \mathrm{id} \in \mathbb{1} + X \leftarrow \mathbb{1} + X$
- $\mathbb{1} + Y \leftarrow \mathbb{1} + X$ to decouple domain and co-domain
- $M.Y \leftarrow M.X$ to allow accumulation of other 'effects'
  M is parameterized (polymorphic) type: Monad with Kleisli composition $\bullet$
- M is functor: also maps functions, and adheres to functor laws
- Monad laws: Kleisli composition $\bullet$ is associative and has unit u