Category Theory as An Excuse to Learn Type Theory

(talk @ Encontro de Categorias 2021)

http://angg.twu.net/math-b.html#2021-excuse-tt

By:
Eduardo Ochs →
(main author)

Selana Ochs →
(friend and coauthor)

Rings

A ring R is a tuple $(R, 0, 1, +, -, \cdot)$ in which:

$$\begin{array}{lll} R & \text{is} & \text{a set}, \\ 0 & \in & R, \\ 1 & \in & R, \\ + & : & R \times R \to R, \\ - & : & R \to R, \\ \cdot & : & R \times R \to R \end{array}$$

and this $(R, 0, 1, +, -, \cdot)$ obeys several conditions.
The part $(R, 0, 1, +, -, \cdot)$ is called structure,
and the conditions are called properties.

(For more on structure and properties
see[FavC, section 7.5], [BS07, p.15]).

Rings (2)

The symbols $0, 1, +, -, \cdot$ in $(R, 0, 1, +, -, \cdot)$ have standard conventions that let us infer their arities, types, and equations — what they have to obey to "deserve their names".

Also, if $f : R \to S$ is a morphism of rings then f is a function from the underlying set of R to the underlying set of S that obeys several equations — for example $f(a + b) = f(a) + f(b)$, that is actually

$$\forall a, b \in R.\, f(a +_R b) = f(a) +_S f(b) \ldots$$

These things can be written in several levels of detail.

Conventions in Category Theory

Category Theory is like that too, only much, much worse.
The notation changes a lot from one text to the other.
Most diagrams are left to the reader.
Some parts of the structure are left to the reader too...
For example (see [FavC, section 3]):

Fix a set A. Let $(A\times)$ be the functor that takes each object B to $A \times B$.

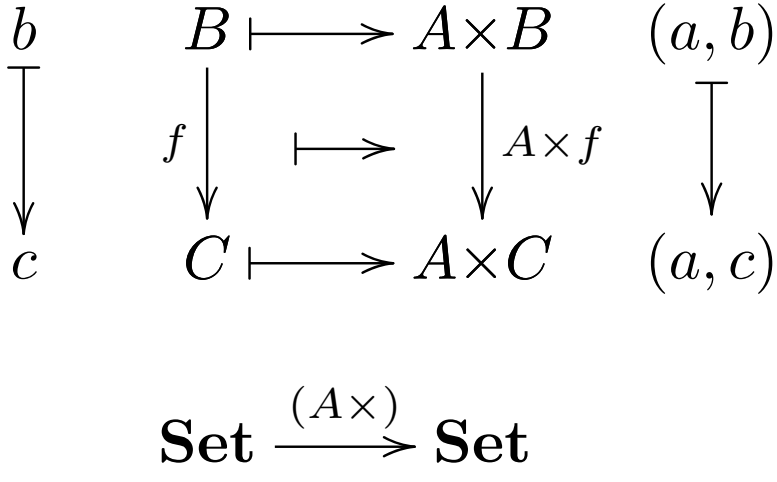
This says that the action on objects of the functor $(A\times)$ is $(A\times)_0(B) = A \times B$, and the "the" implies that the reader knows how to discover that the action on morphisms is:

$$(A\times)_1(B \xrightarrow{f} C) = A \times B \xrightarrow{\langle \pi, f\circ\pi' \rangle} A \times C.$$

A weird idea: downcasing

This looked very natural to me when I was starting to learn CT and Haskell by myself... it is great as an informal tool.

Btw, the paper [IDARCT] is about how I tried to formalize this idea for years, failed miserably, and then found a way to use it for "skeletons of proofs".



$$
\begin{aligned}
&(a,b) \mapsto (a,c)\\
&\equiv (a,b) \mapsto (a, f(b))\\
&\equiv (a,b) \mapsto (\pi(a,b), f(\pi'(a,b)))\\
&\equiv p \mapsto (\pi p, f(\pi' p))\\
&\equiv p \mapsto (\pi(p), (f \circ \pi')(p))\\
&\equiv p \mapsto \langle \pi, f \circ \pi' \rangle (p)\\
&\equiv \langle \pi, f \circ \pi' \rangle
\end{aligned}
$$

A weird idea: downcasing (2)

Downcasing and upcasing are like
lowercasing and uppercasing, but for terms and types.

Downcasing a set gives us a name for a
"typical element" of that set.
Downcasing a type gives us a name for a variable of that type.

a	:	A	(ok)
a'	:	A'	(ok)
a'	:	A	(ok when there's no A')
b	:	B	(ok)
(a, b)	:	$A \times B$	(ok?)
$b \mapsto c$	:	$B \to C$	(weird)
$(a, b) \mapsto (a, c)$	:	$A \times B \to A \times C$	(weird)

Term inference is like proof search

Finding the meaning of $A \times f$
means finding a natural construction
for $? : A \times B \to A \times C$ from $f : B \to C$...

"Natural constructions" are λ-terms.
This is a bit quirky. Details: [FavC, section 3].
Main diagrams:

$$\frac{b \mapsto c}{(a,b) \mapsto (a,c)} \quad \rightsquigarrow \quad \frac{\dfrac{\dfrac{[(a,b)]^1}{a} \qquad \dfrac{\dfrac{[(a,b)]^1}{b} \qquad b \mapsto c}{c}}{(a,c)}}{(a,b) \mapsto (a,c)}\,1 \quad \rightsquigarrow \quad \frac{\dfrac{\dfrac{[p]^1}{\pi p} \qquad \dfrac{\dfrac{p}{\pi' p} \qquad f}{f(\pi' p)}}{(\pi p, f(\pi' p))}}{\lambda p.(\pi p, f(\pi' p))}\,1$$

$$\frac{Q \to R}{P \wedge Q \to P \wedge R} \quad \rightsquigarrow \quad \frac{\dfrac{\dfrac{[P \wedge Q]^1}{P} \qquad \dfrac{\dfrac{[P \wedge Q]^1}{Q} \qquad Q \to R}{R}}{P \wedge R}}{P \wedge Q \to P \wedge R}\ 1$$

$$\frac{f : B \to C}{(\times A)_1 f : A \times C \to A \times C} \quad \rightsquigarrow \quad \frac{\dfrac{\dfrac{[p : A \times B]^1}{\pi p : A} \qquad \dfrac{\dfrac{[p : A \times B]^1}{\pi' p : B} \qquad f : B \to C}{f(\pi' p) : C}}{(\pi p, f(\pi' p)) : A \times C}}{(\lambda p{:}A \times B.(\pi p, f(\pi' p)) : A \times B \to A \times C}\ 1$$

$$\dfrac{\dfrac{\dfrac{[p : A{\times}B]^1}{\pi p : A} \qquad \dfrac{\dfrac{[p : A{\times}B]^1}{\pi' p : B} \qquad f : B \to C}{f(\pi' p) : C}}{(\pi p, f(\pi' p)) : A{\times}C}}{(\lambda p{:}A{\times}B.(\pi p, f(\pi' p)) : A{\times}B \to A{\times}C}\ 1$$

$$\dfrac{\dfrac{\dfrac{[\texttt{ab} : A{\times}B]^1}{\texttt{a} : A} \qquad \dfrac{\dfrac{[\texttt{ab} : A{\times}B]^1}{\texttt{b} : B} \qquad \texttt{btoc} : B \to C}{\texttt{c} : C}}{\texttt{ac} : A{\times}C}}{\texttt{abtoac} : A{\times}B \to A{\times}C}\ 1$$

```
a = fst ab
b = snd ab
c = btoc b
ac = (a,c)
abtoac = \ab -> ac
atimes = \btoc -> abtoac
```

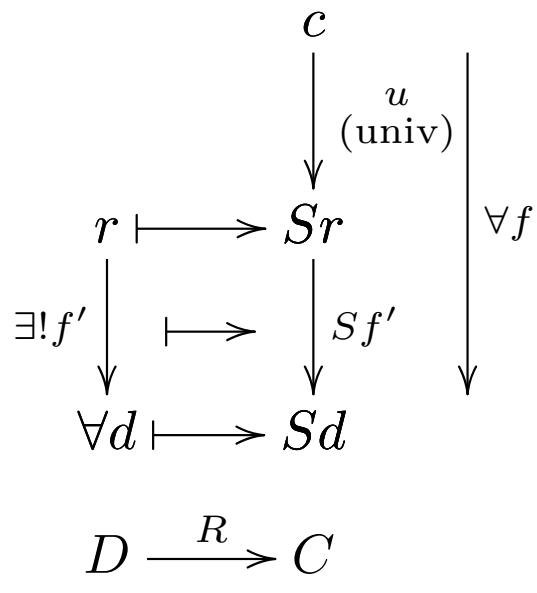
```
-- Translation to Haskell:
-- (with help from ski @ #haskell)
atimes :: (b -> c) -> ((a,b) -> (a,c))
atimes btoc = abtoac where
  abtoac ab = ac where
    a = fst ab
    b = snd ab
    c = btoc b
    ac = (a,c)
```

Here's how [CWM, section III.1] defines universals:

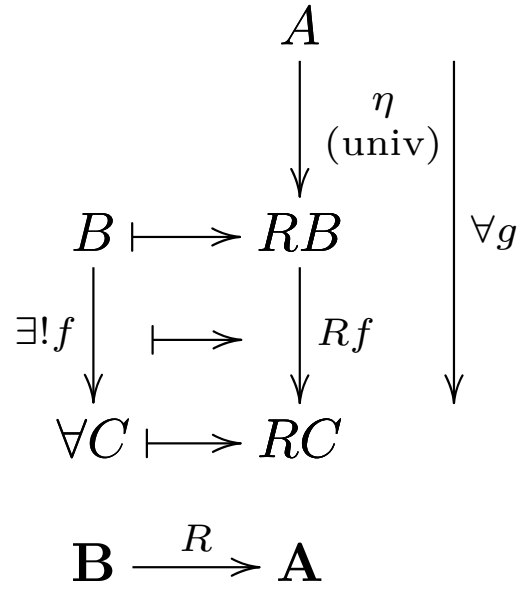
> **Definition.** If $S : D \to C$ is a functor and c an object of C, a universal arrow from c to S is a pair $\langle r, u \rangle$ consisting of an object r of D and an arrow $u : c \to Sr$ of C, such that to every pair $\langle d, f \rangle$ with d an object of D and $f : c \to Sd$ an arrow of C, there is a unique arrow $f' : r \to d$ of D with $Sf' \circ u = f$.

This is too hard for my tiny brain.
Solution: I need types, diagrams,
a way to use other letters and fonts,
and examples.

If C, D are categories,
$S : D \to C$ is a functor,
$c \in C$ is an object,
$r \in D$ is an object,
$u : C \to Sr$ is a morphism,
then $\langle r, u \rangle$ is universal from c to S
iff $\forall d \in D$,
$\forall f : C \to Sd$,
$\exists! f : r \to d$ with $Sf' \circ u = f$

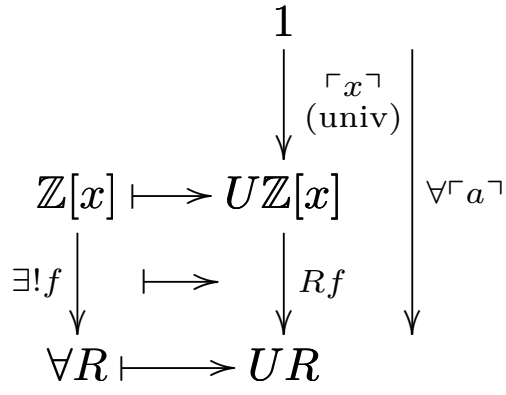


If $\mathbf{A}, \mathbf{B}$ are categories,
$R : \mathbf{A} \to \mathbf{B}$ is a functor,
$A \in \mathbf{A}$ is an object,
$B \in \mathbf{B}$ is an object,
$\eta : A \to RB$ is a morphism,
then (B, η) is universal from A to R
iff $\forall C \in \mathbf{B}$,
$\forall g : A \to RC$,
$\exists! f : B \to C$ with $Rf \circ \eta = g$



↑ Looks like
Type Theory

↑ Looks like
Freyd's notation

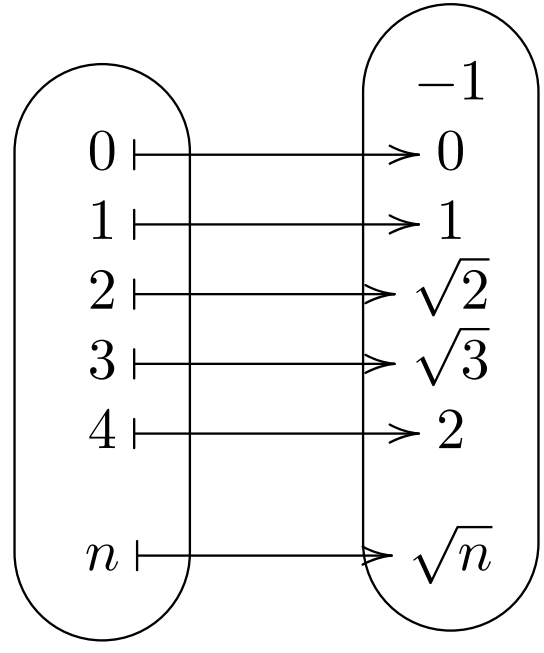


If	**Set**, **Rings** are categories,	(ok)
	U : **Rings** $\to$ **Set** is a functor,	(ok: U = underlying set)
	$1 \in$ **Set** is an object,	(ok: $1 = \{*\}$)
	$\mathbb{Z}[x] \in$ **Rings** is an object,	(ok)
	$\ulcorner x \urcorner : 1 \to U\mathbb{Z}[x]$ is a morphism,	(ok: $\ulcorner x \urcorner(*) = x$)
then	$(\mathbb{Z}[x], \ulcorner x \urcorner)$ is universal from A to R	
iff	$\forall R \in$ **Rings**,	
	$\forall \ulcorner a \urcorner : 1 \to UR$,	
	$\exists! f : \mathbb{Z}[x] \to R$ with $Uf \circ \ulcorner x \urcorner = \ulcorner a \urcorner$	(ok: $f(P(x)) = P(a)$)

Internal views

The internal view of a function:

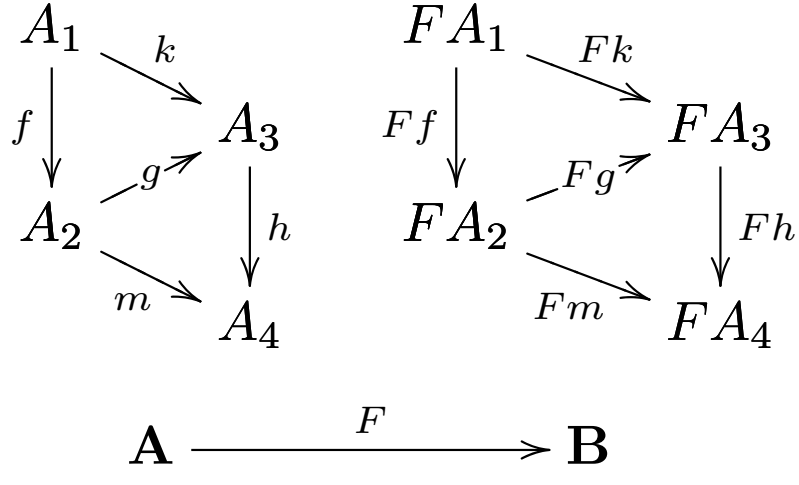
$$\begin{array}{rccc} \sqrt{}: & \mathbb{N} & \to & \mathbb{R} \\ & n & \mapsto & \sqrt{n} \end{array}$$



$$\mathbb{N} \xrightarrow{\sqrt{}} \mathbb{R}$$

The internal view of a functor

Above usually means *inside*. (CAI)
The image of a diagram above $\mathbf{A}$ by a functor $F : \mathbf{A} \to \mathbf{B}$ is a a diagram *with the same shape* above $\mathbf{B}$. (CFSh)



In this case we don't draw the arrows like $A_1 \mapsto FA_1$ because there would be too many of them — we leave them implicit.

The internal view of a functor (2)

We say that the diagram in the previous slide is an internal view of the functor F. To draw the internal view of the functor $F : \mathbf{A} \to \mathbf{B}$ we start with a diagram in $\mathbf{A}$ that is made of two generic objects and a generic morphism between them. We get this:

$$\begin{array}{ccc} C & \longmapsto & FC \\ {\scriptstyle g}\downarrow & \longmapsto & \downarrow{\scriptstyle Fg} \\ D & \longmapsto & FD \end{array}$$

$$\mathbf{A} \xrightarrow{F} \mathbf{B}$$

The internal view of a functor (3)

Any arrow of the form $\alpha \mapsto \beta$ above a functor arrow $\mathbf{A} \xrightarrow{F} \mathbf{B}$ is interpreted as saying that F takes α to β, or, that $F\alpha$ reduces to β. So this diagram

$$\begin{array}{ccc} B & \longmapsto & A \times B \\ {\scriptstyle f}\downarrow & \longmapsto & \downarrow{\scriptstyle \lambda p.(\pi p, f(\pi' p))} \\ C & \longmapsto & A \times C \end{array}$$

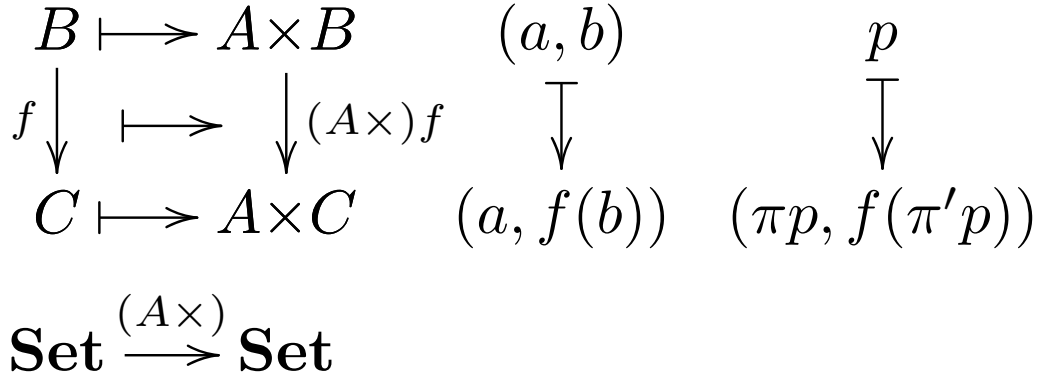
$$\mathbf{Set} \xrightarrow{(A\times)} \mathbf{Set}$$

defines $(A\times)$ as:

$$\begin{array}{lcl} (A\times)_0 & := & \lambda B.\, A \times B, \\ (A\times)_1 & := & \lambda f.\lambda p.(\pi p, f(\pi' p)). \end{array}$$

The internal view of a functor (4)

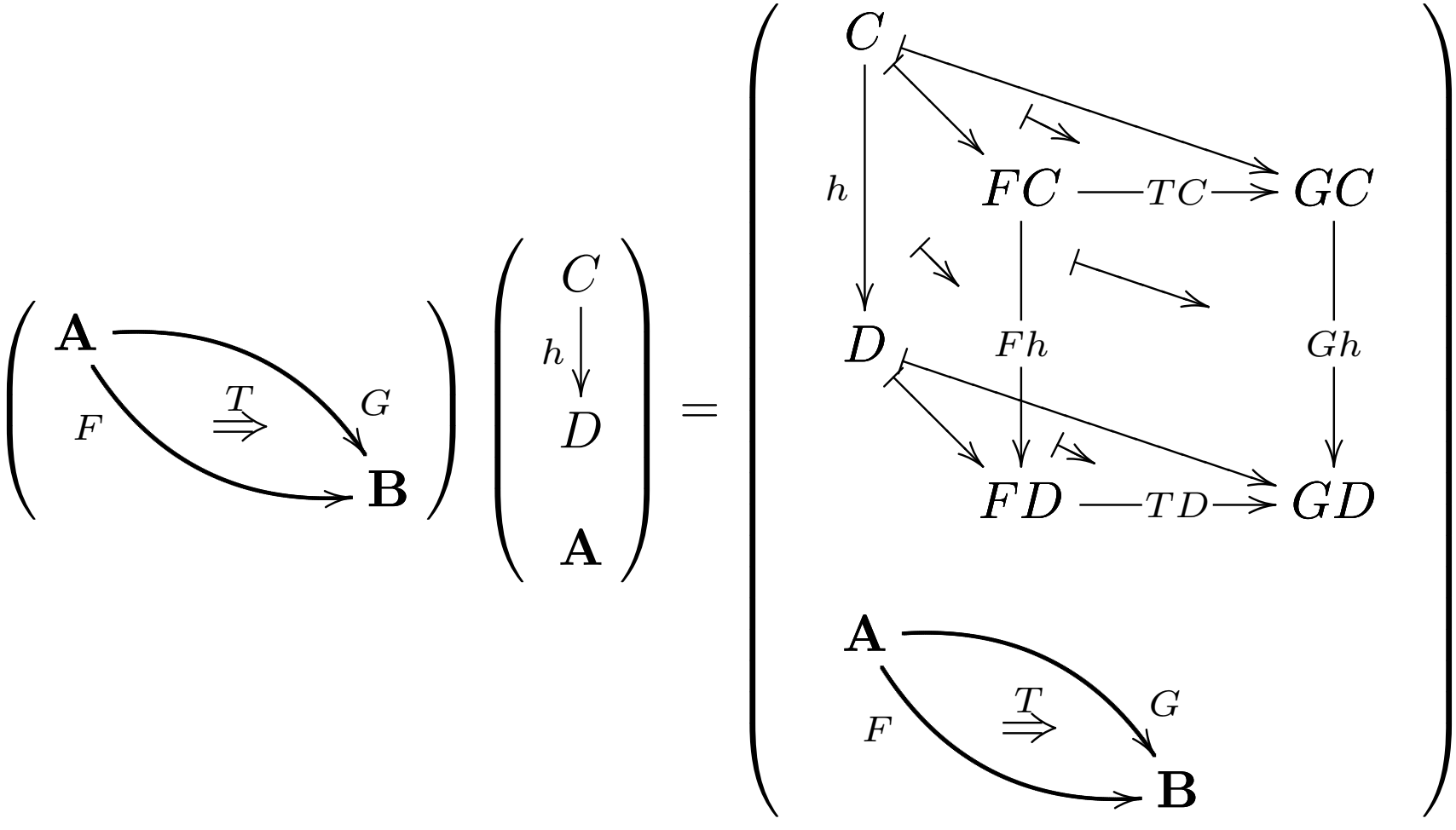
Here the codomain of F is **Set**, and so we can also use the internal view of $(A\times)f$ to define $(A\times)_1$...

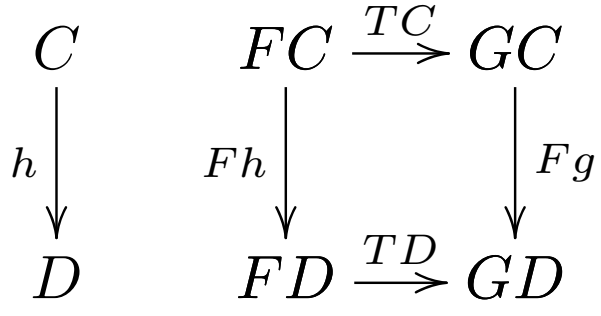


How to draw the internal view of a NT

If $F, G : \mathbf{A} \to \mathbf{B}$ are functors from $\mathbf{A}$ to $\mathbf{B}$,
$T : F \to G$ is a natural transformation from F to G, and
$h : C \to D$ is a morphism in $\mathbf{A}$,
then Th is a commutative square in $\mathbf{B}$.

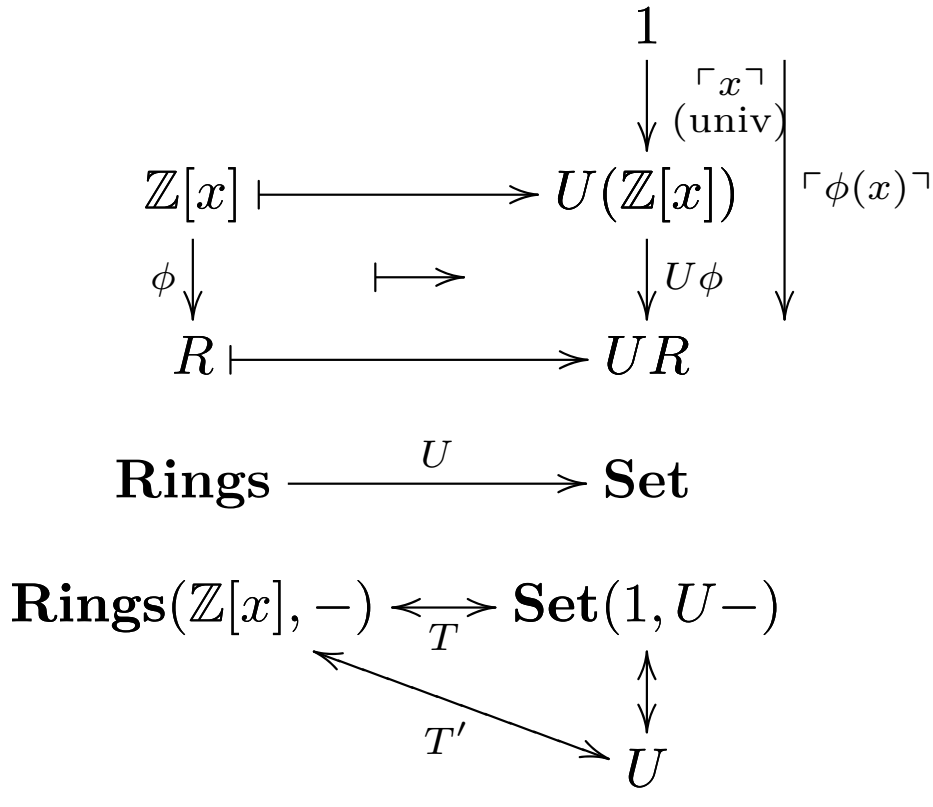
How do we visualise this?
How do we visualise this with all details?


A
F
T
G
B
C
h
D
A
=
C
h
D
FC
TC
GC
Fh
Gh
FD
TD
GD
A
F
T
G
B

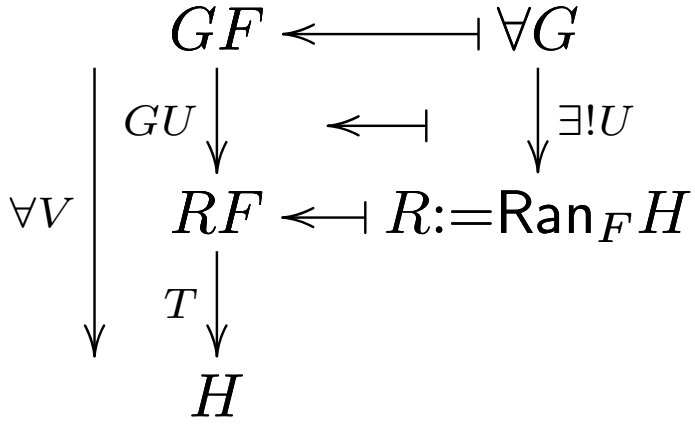


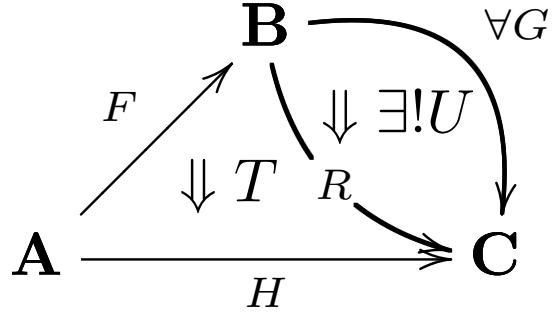
$$\mathbf{A} \xrightarrow{T} \mathbf{B}$$

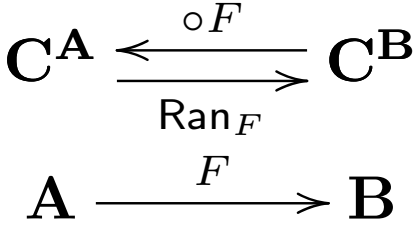
Trailer: Yoneda in a particular case
(With a representable functor... [FavC, section 7.7])



Trailer: (right) Kan Extensions
(From [FavC, section 7.9])



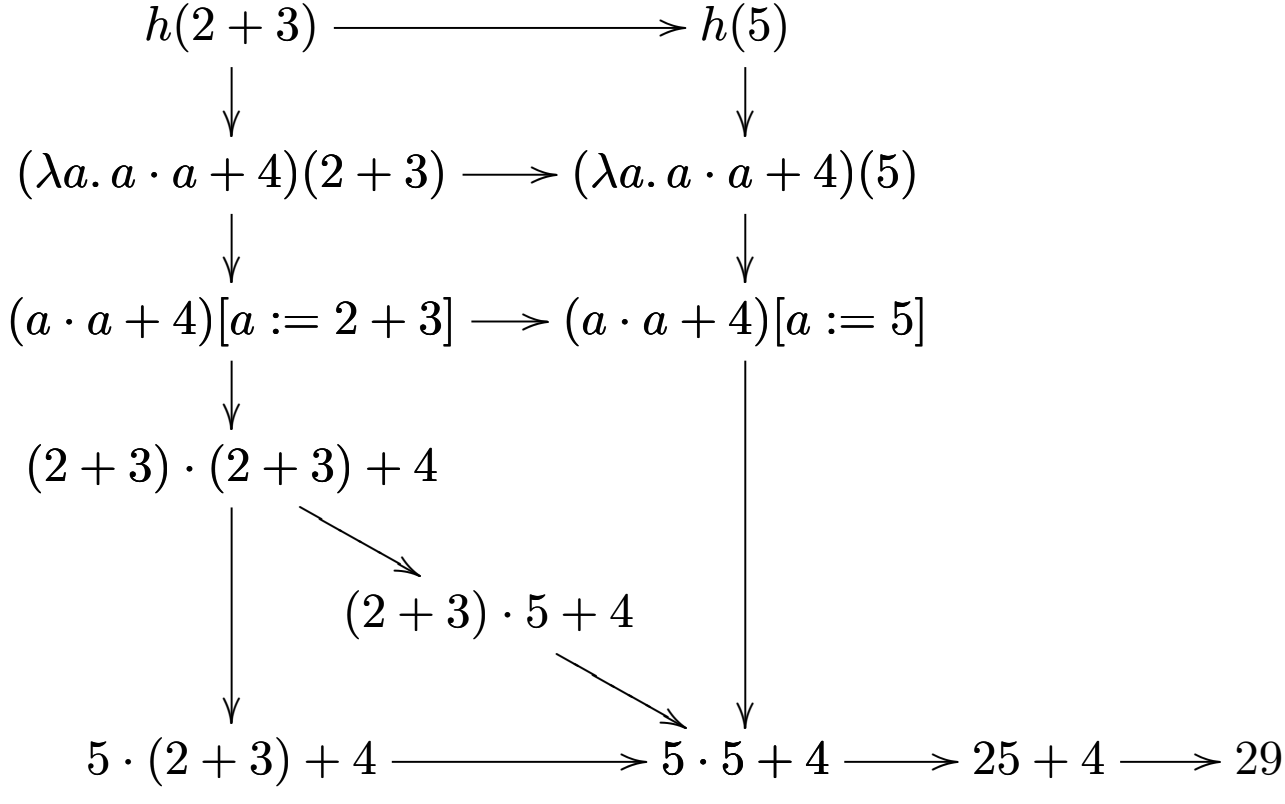




Introduction to Type Theory

A bit of untyped λ-calculus

If $h = (\lambda a.a \cdot a + 4)$ then...



Types after Discrete Mathematics

If:

$A = \{1, 2\},$
$B = \{3, 4\},$
$C = \{30, 40\},$
$D = \{10, 20\},$

Then:

$$A \times B = \begin{Bmatrix} (1,3), (1,4), \\ (2,3), (2,4) \end{Bmatrix}$$

$$B \to C = \left\{ \left\{ \begin{matrix} (3,30), \\ (4,30) \end{matrix} \right\}, \left\{ \begin{matrix} (3,30), \\ (4,40) \end{matrix} \right\}, \left\{ \begin{matrix} (3,40), \\ (4,30) \end{matrix} \right\}, \left\{ \begin{matrix} (3,40), \\ (4,40) \end{matrix} \right\} \right\}$$

If we know (the values of) a, b, f
then we know (the value of) $(a, f(b))$.
If $(a, b) = (2, 3)$ and $f = \left\{ \begin{smallmatrix} (3,30), \\ (4,40) \end{smallmatrix} \right\}$
then $(a, f(b)) = (2, 30)$.

In trees:

$$\dfrac{\dfrac{(a,b)}{a}\,\pi \quad \dfrac{\dfrac{(a,b)}{b}\,\pi' \quad f}{f(b)}\,\mathsf{app}}{(a, f(b))}\,\mathsf{pair} \qquad \dfrac{\dfrac{(2,3)}{2}\,\pi \quad \dfrac{\dfrac{(2,3)}{3}\,\pi' \quad \left\{ \begin{smallmatrix} (3,30), \\ (4,40) \end{smallmatrix} \right\}}{30}\,\mathsf{app}}{(2, 30)}\,\mathsf{pair}$$

If we know the *types* of a, b, f
we know the type of $(a, f(b))$.
If we know the types of p, f
we know the type of $(\pi p, f(\pi' p))$.
If we know the types of p, f
we know the type of $(\lambda p : A \times B.(\pi p, f(\pi' p)))$.

$$\dfrac{\dfrac{(a,b) : A \times B}{a : A}\ \pi \qquad \dfrac{\dfrac{(a,b) : A \times B}{b : B}\ \pi' \qquad f : B \to C}{f(b) : C}\ \mathsf{app}}{(a, f(b)) : A \times C}\ \mathsf{pair}$$

$$\dfrac{\dfrac{\dfrac{p : A \times B}{\pi p : A}\ \pi \qquad \dfrac{\dfrac{p : A \times B}{\pi' p : B}\ \pi' \qquad f : B \to C}{f(\pi' p) : C}\ \mathsf{app}}{(\pi p, f(\pi' p)) : A \times C}\ \mathsf{pair}}{(\lambda p : A \times B.(\pi p, f(\pi' p))) : A \times B \to A \times C}\ \lambda$$

Compare:

$$\dfrac{\dfrac{\dfrac{p : A \times B}{\pi p : A}\,\pi \qquad \dfrac{\dfrac{p : A \times B}{\pi' p : B}\,\pi' \qquad f : B \to C}{f(\pi' p) : C}\,\mathsf{app}}{(\pi p, f(\pi' p)) : A \times C}\,\mathsf{pair}}{(\lambda p : A \times B.(\pi p, f(\pi' p))) : A \times B \to A \times C}\,\lambda$$

$$\lambda \underbrace{p}_{:A\times B} : A \times B.\,\underbrace{\underbrace{(\pi \underbrace{p}_{:A\times B}}_{:A}, \underbrace{\underbrace{f}_{:B\to C}\,\underbrace{(\pi' \underbrace{p}_{:A\times B})}_{:B}}_{:C}}_{:A\times C}}_{:A\times B\to A\times C}$$

Typing

Idea: start with lots of type variables,
look at the constraints induced by the underbraces,
try to find a substitution that works...

$$\underbrace{\underbrace{\lambda\ \underbrace{p}_{:A}\ .\ (\pi\ \underbrace{p}_{:A}\ ,\ \underbrace{f}_{:B}\ (\pi'\ \underbrace{p}_{:A}\ ))}_{:F}}_{:G}$$

$$\begin{array}{ll} (C \Rightarrow) & A = C \times \_ \\ (D \Rightarrow) & A = \_ \times D \\ (E \Rightarrow) & B = D \to E \\ (F \Rightarrow) & F = C \times E \\ (G \Rightarrow) & G = A \to F \end{array}$$

First step: $[A := C \times D]$

In the previous slide all the constraints
were red, meaning “not satisfied yet”.
Let’s mark the ones that are satisfied in green
and perform more substituions if needed...

$$\underbrace{\lambda \underbrace{p}_{:C\times D} . \underbrace{(\pi \underbrace{p}_{:C\times D}}_{:C}, \underbrace{\underbrace{f}_{:B} \underbrace{(\pi' \underbrace{p}_{:C\times D}}_{:D})}_{:E}}_{:F})}_{:G}$$

$$\begin{aligned}
(C \Rightarrow) \quad & C \times D = C \times _ \\
(D \Rightarrow) \quad & C \times D = _ \times D \\
(E \Rightarrow) \quad & B = D \to E \\
(F \Rightarrow) \quad & F = C \times E \\
(G \Rightarrow) \quad & G = C \times D \to F
\end{aligned}$$

Next step: $\begin{bmatrix} B := D \to E \\ F := C \times E \end{bmatrix}$

Let's
do
it
again...

$$\underbrace{\lambda \underbrace{p}_{:C\times D}.\underbrace{(\pi \underbrace{p}_{:C\times D}}_{:C}, \underbrace{\underbrace{f}_{:D\to E} \underbrace{(\pi' \underbrace{p}_{:C\times D}}_{:D})}_{:E})}_{:C\times E})}_{:G}$$

$$\begin{array}{ll} (C \Rightarrow) & C \times D = C \times \_ \\ (D \Rightarrow) & C \times D = \_ \times D \\ (E \Rightarrow) & D \to E = D \to E \\ (F \Rightarrow) & C \times E = C \times E \\ (G \Rightarrow) & G = C \times D \to C \times E \end{array}$$

Next step: $[G := C \times D \to C \times E]$

Done! Note that:
1) C, D, E are *type variables*,
2) this typing is *universal* — all other valid typing are substitution instances of this one. For example...
(see the next page)

Exercise:
Compare our naïve approach here
with the ones in [Pie02, chapter 22].

$$\left(\lambda \underbrace{p}_{:C\times D} . (\pi \underbrace{\underbrace{p}_{:C\times D}}_{:C}, \underbrace{f}_{:D\to E} \underbrace{(\pi' \underbrace{p}_{:C\times D})}_{:D})\right)_{:C\times D\to C\times E} \left[\begin{matrix} C := A \\ D := B \\ E := C \end{matrix}\right] =$$

$$\left(\lambda \underbrace{p}_{:A\times B} . (\pi \underbrace{\underbrace{p}_{:A\times B}}_{:A}, \underbrace{f}_{:B\to C} \underbrace{(\pi' \underbrace{p}_{:A\times B})}_{:B})\right)_{:A\times B\to A\times C}$$

Very very concrete types

$$\left(\underbrace{\lambda\ \underbrace{p}_{:A\times B}\ :A\times B.\left(\pi\ \underbrace{\underbrace{p}_{:A\times B}}_{:A},\ \underbrace{f}_{:B\to C}\ \underbrace{\left(\pi'\ \underbrace{p}_{:A\times B}\right)}_{:B}\right)}_{:C}}_{:A\times C}\right)_{:A\times B\to A\times C}\left[\begin{matrix}A:=\{0,1\}\\B:=\{2,3\}\\C:=\{4,5\}\end{matrix}\right]=$$

$$\left(\underbrace{\lambda\ \underbrace{p}_{:\{0,1\}\times\{2,3\}}\ :\{0,1\}\times\{2,3\}.\left(\pi\ \underbrace{\underbrace{p}_{:\{0,1\}\times\{2,3\}}}_{:\{0,1\}},\ \underbrace{f}_{:\{2,3\}\to\{4,5\}}\ \underbrace{\left(\pi'\ \underbrace{p}_{:\{0,1\}\times\{2,3\}}\right)}_{:\{2,3\}}\right)}_{:\{4,5\}}}_{:\{0,1\}\times\{4,5\}}\right)_{:\{0,1\}\times\{2,3\}\to\{0,1\}\times\{4,5\}}$$

Introduction to the Hard Part of Basic Type Theory

Girard:
[GLT89] and [Gir10]

Benjamin Pierce's book, Haskell:
[Pie02], [Zav20], [Hut16]

HOTT, Kamareddine:
[HOTT], [KLN04]

Also:
[Som00], [BHL20]

Some of them don't even mention Σ-types...
"Σ-types can be constructed from Π-types"
This is not trivial at all! =(

Contexts

$$\{\underbrace{\underbrace{a \in \{1,2\}}_{\text{gen}}, \underbrace{b \in \{2,3\}}_{\text{gen}}, \underbrace{a < b}_{\text{filt}}}_{\text{context}} ; \underbrace{10a + b}_{\text{expr}}\}$$

here the context has "generators" ("*var* $\in$ *set*")
and "filters" ("*this expression must be true*").

In

$$\underbrace{\underbrace{A{:}\Theta}_{\text{v:T}}, \underbrace{B{:}\Theta}_{\text{v:T}}, \underbrace{C{:}\Theta}_{\text{v:T}}, \underbrace{p : A \times B}_{\text{v:T}}, \underbrace{g : B \to C}_{\text{v:T}}}_{\text{context}} \vdash \underbrace{\underbrace{(\pi p, g(\pi' p))}_{\text{term}} : \underbrace{A \times C}_{\text{type}}}_{\text{conclusion}}$$

the context is made of several declarations of the form
"variable : type"...

Derivations in Pure Type Systems

Derivations in a PTS have *lots* of bureaucracy.

This is what we need to derive $A{:}\Theta, B{:}\Theta \vdash (\Pi a{:}A.B){:}\Theta$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \Theta{:}\Box}\,a \quad \dfrac{}{\vdash \Theta{:}\Box}\,a}{A{:}\Theta \vdash \Theta{:}\Box}\,w_\Box \quad \dfrac{}{\vdash \Theta{:}\Box}\,a}{A{:}\Theta \vdash A{:}\Theta}\,v_\Box}{A{:}\Theta, B{:}\Theta \vdash A{:}\Theta}\,w_\Box \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \Theta{:}\Box}\,a \quad \dfrac{}{\vdash \Theta{:}\Box}\,a}{A{:}\Theta \vdash \Theta{:}\Box}\,w_\Box \quad \dfrac{}{\vdash \Theta{:}\Box}\,a}{A{:}\Theta \vdash A{:}\Theta}\,v_\Box}{A{:}\Theta, B{:}\Theta \vdash A{:}\Theta}\,w_\Box \quad \dfrac{\dfrac{\dfrac{}{\vdash \Theta{:}\Box}\,a \quad \dfrac{}{\vdash \Theta{:}\Box}\,a}{A{:}\Theta \vdash \Theta{:}\Box}\,w_\Box}{A{:}\Theta, B{:}\Theta \vdash B{:}\Theta}\,v_\Box}{A{:}\Theta, B{:}\Theta, a{:}A \vdash B{:}\Theta}\,w_\Theta}{A{:}\Theta, B{:}\Theta \vdash (\Pi a{:}A.B){:}\Theta}\,\Pi_{\Theta\Theta\Theta}$$

i.e.:

$$\dfrac{\dfrac{\dfrac{}{A{:}\Theta, B{:}\Theta \vdash A{:}\Theta} \qquad \dfrac{\dfrac{\dfrac{}{A{:}\Theta \vdash \Theta{:}\Box} \quad \dfrac{}{\vdash \Theta{:}\Box}\,a}{A{:}\Theta \vdash A{:}\Theta}\,v_\Box}{A{:}\Theta, B{:}\Theta \vdash A{:}\Theta}\,w_\Box \qquad \dfrac{\dfrac{\dfrac{}{\vdash \Theta{:}\Box}\,a \quad \dfrac{}{\vdash \Theta{:}\Box}\,a}{A{:}\Theta \vdash \Theta{:}\Box}\,w_\Box}{A{:}\Theta, B{:}\Theta \vdash B{:}\Theta}\,v_\Box}{A{:}\Theta, B{:}\Theta, a{:}A \vdash B{:}\Theta}\,w_\Theta}{A{:}\Theta, B{:}\Theta \vdash (\Pi a{:}A.B){:}\Theta}\,\Pi_{\Theta\Theta\Theta}$$

Pure Type Systems: a derivation (2)

This is what we need to derive $A{:}\Theta \vdash (\lambda a{:}A.a){:}(\Pi a{:}A.A){:}$

$$\dfrac{\dfrac{}{A{:}\Theta \vdash A{:}\Theta}\,v_\Theta}{A{:}\Theta, a{:}A \vdash a{:}A}\qquad \dfrac{\dfrac{}{A{:}\Theta \vdash A{:}\Theta}\qquad \dfrac{\dfrac{\dfrac{}{A{:}\Theta \vdash A{:}\Theta}}{A{:}\Theta \vdash A{:}\Theta}\,v_\Box}{A{:}\Theta, a{:}A \vdash A{:}\Theta}\,w_\Theta}{A{:}\Theta \vdash (\Pi a{:}A.A){:}\Theta}\,\Pi_{\Theta\Theta\Theta}}{A{:}\Theta \vdash (\lambda a{:}A.a){:}(\Pi a{:}A.A)}\,\lambda \qquad \dfrac{}{\vdash \Theta{:}\Box}\,a$$

Exercise:
Read the presentation of PTSs in [KLN04, section 4c].
Translate these trees to the notation of the book.
You will need a few tiny changes.

Judgments

If we think — informally — that
Θ is the "set of all sets" (mnemonic: Theta for "Theths"),
then we can put these judgments for λ-calculus in
a very homogeneous form...

$$\underbrace{\underbrace{A{:}\Theta}_{\text{v:T}},\underbrace{B{:}\Theta}_{\text{v:T}},\underbrace{C{:}\Theta}_{\text{v:T}},\underbrace{p : A \times B}_{\text{v:T}},\underbrace{g : B \to C}_{\text{v:T}}}_{\text{context}} \vdash \underbrace{\underbrace{(\pi p, g(\pi' p))}_{\text{term}} : \underbrace{A \times C}_{\text{type}}}_{\text{conclusion}}$$

the context is a series of declarations of the form "var : type",
and the conclusion is of the form "term : type".

Dependent types for children

$A = \{0, 1\}$,
$B = \{2, 3\}$,
$C_0 = \{4, 5\}$,
$C_1 = \{6, 7\}$

$f : A \to B$
$f(a) \in B$

$g : (a{:}A) \to C_a$
$g(a) \in C_a$

$p \in A \times B$
$\pi' p \in B$

$q \in (a{:}A) \times C_a$
$\pi' q \in C_{\pi q}$

Function types and
dependent function types (Π-types):

$$A \to B = \left\{ \left\{ \begin{smallmatrix}(0,2),\\(1,2)\end{smallmatrix} \right\}, \left\{ \begin{smallmatrix}(0,2),\\(1,3)\end{smallmatrix} \right\}, \left\{ \begin{smallmatrix}(0,3),\\(1,2)\end{smallmatrix} \right\}, \left\{ \begin{smallmatrix}(0,4),\\(1,4)\end{smallmatrix} \right\} \right\}$$

$$(a : A) \to C_a =$$
$$\Pi a : A.C_a = \left\{ \left\{ \begin{smallmatrix}(0,4),\\(1,6)\end{smallmatrix} \right\}, \left\{ \begin{smallmatrix}(0,4),\\(1,7)\end{smallmatrix} \right\}, \left\{ \begin{smallmatrix}(0,5),\\(1,6)\end{smallmatrix} \right\}, \left\{ \begin{smallmatrix}(0,5),\\(1,7)\end{smallmatrix} \right\} \right\}$$

Product types and
dependent pair types (Σ-types):

$$A \times B = \{(0,2),\ (0,3),\ (1,2),\ (1,3)\}$$

$$(a : A) \times C_a =$$
$$\Sigma a : A.C_a = \{(0,4),\ (0,5),\ (1,6),\ (1,7)\}$$

Note: "$(a : A) \to C_a$" and "$(a : A) \times C_a$" are Agda-isms.
Many places use the terms "dependent products" and "dependent sums".

Typing proofs of "and" and "implies"

$\langle\langle Q \rangle\rangle$ is a proof of Q.
$[\![Q]\!]$ is the set of all proofs of Q.

$[\![Q \wedge R]\!]$ is the set of all proofs of $Q \wedge R$.
A $\langle\langle Q \wedge R \rangle\rangle$ is a proof of $Q \wedge R$.
A $\langle\langle Q \wedge R \rangle\rangle$ is a pair $(\langle\langle Q \rangle\rangle, \langle\langle R \rangle\rangle)$,
so $[\![Q \wedge R]\!] = [\![Q]\!] \times [\![R]\!]$.

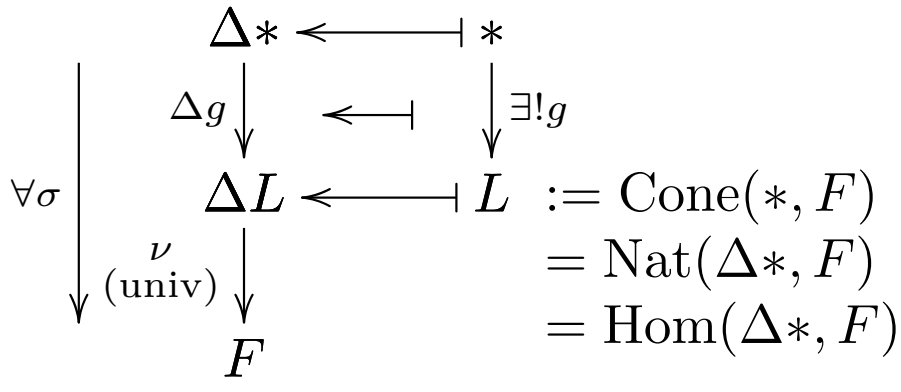
$[\![Q \to R]\!]$ is the set of all proofs of $Q \to R$.
A $\langle\langle Q \to R \rangle\rangle$ is a proof of $Q \to R$.
A $\langle\langle Q \to R \rangle\rangle$ is an operation that takes $\langle\langle Q \rangle\rangle$s to $\langle\langle R \rangle\rangle$s,
so $[\![Q \to R]\!] = [\![Q]\!] \to [\![R]\!]$.

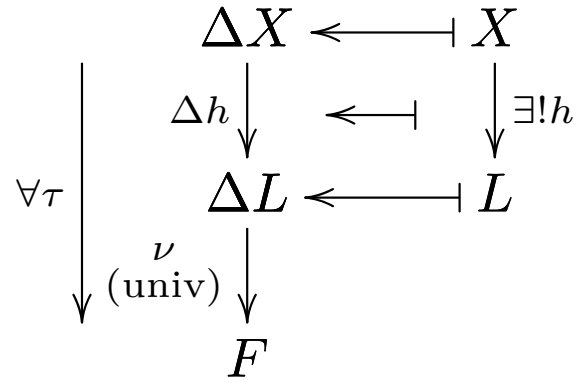
Typing proofs of "for all" and "exists"

$$\underbrace{\langle\!\langle \forall b{:}B.P(b) \rangle\!\rangle}_{:[\![\forall b{:}B.P(b)]\!]} = \underbrace{\left\{ \begin{array}{l} (2, \langle\!\langle P(2) \rangle\!\rangle), \\ (3, \langle\!\langle P(3) \rangle\!\rangle) \end{array} \right\}}_{:\Pi b{:}B.[\![P(b)]\!]}$$

$$\underbrace{\langle\!\langle \exists b{:}B.P(b) \rangle\!\rangle}_{:[\![\exists b{:}B.P(b)]\!]} = \underbrace{(b, \langle\!\langle P(b) \rangle\!\rangle)}_{:\Sigma b{:}B.[\![P(b)]\!]}$$

CWM: Creation of Limits, theorem V.1

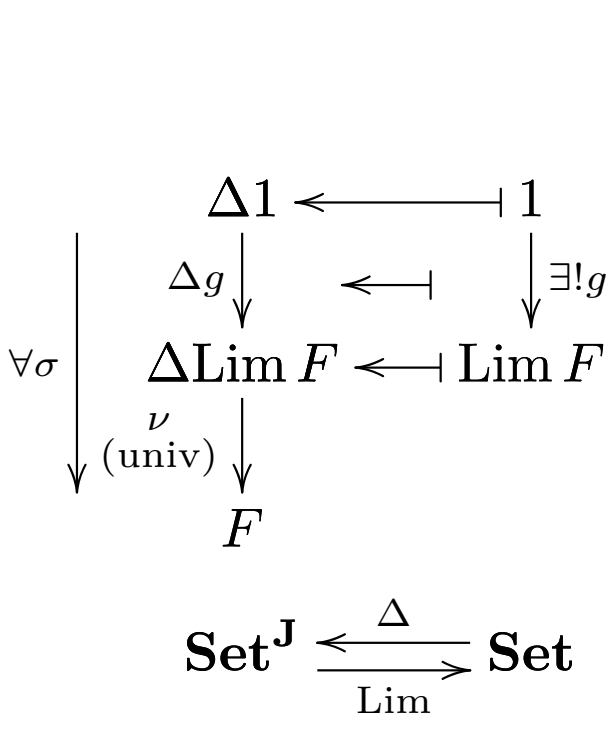


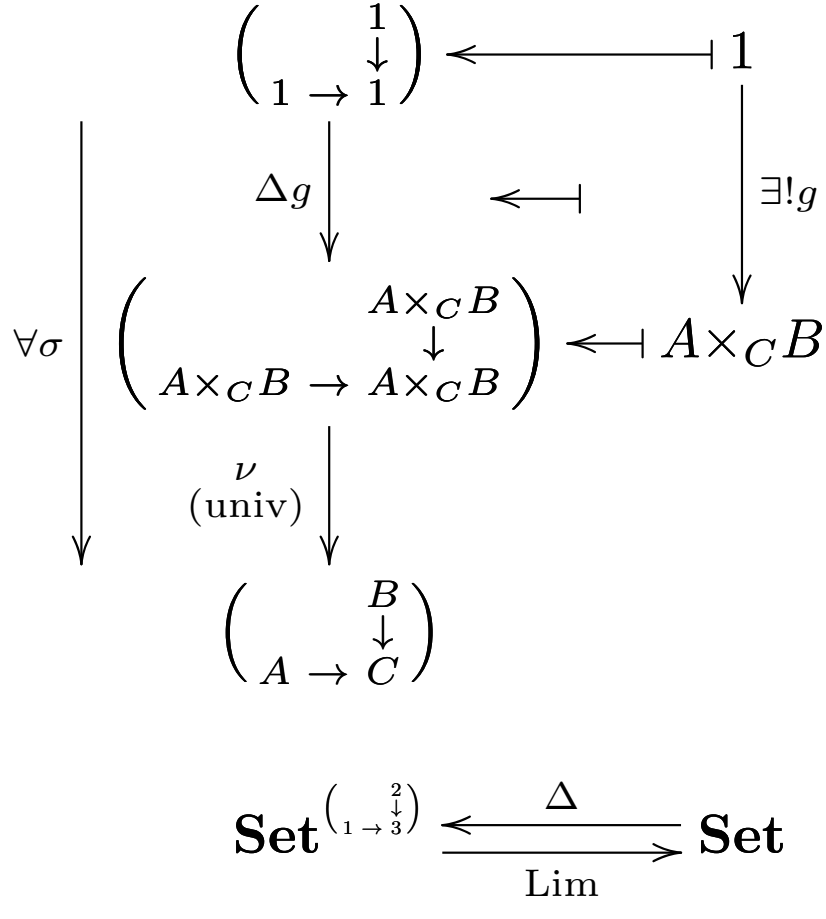


$$\mathrm{Set}^J \underset{\mathrm{Lim}}{\overset{\Delta}{\leftrightarrows}} \mathrm{Set}$$

$$\mathrm{Set}^J \underset{\mathrm{Lim}}{\overset{\Delta}{\leftrightarrows}} \mathrm{Set}$$

Guessing ν: first attempt





In this first attempt the types don't match...
A natural transformation

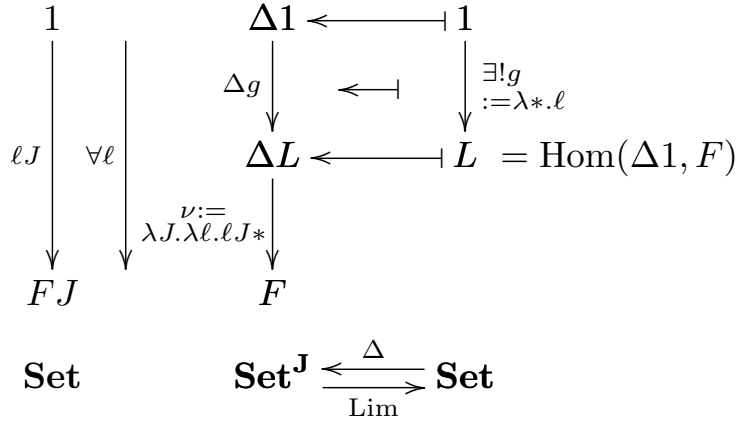
$$\sigma : \begin{pmatrix} & 1 \\ & \downarrow \\ 1 & \to 1 \end{pmatrix} \to \begin{pmatrix} & B \\ & \downarrow \\ A & \to C \end{pmatrix}$$

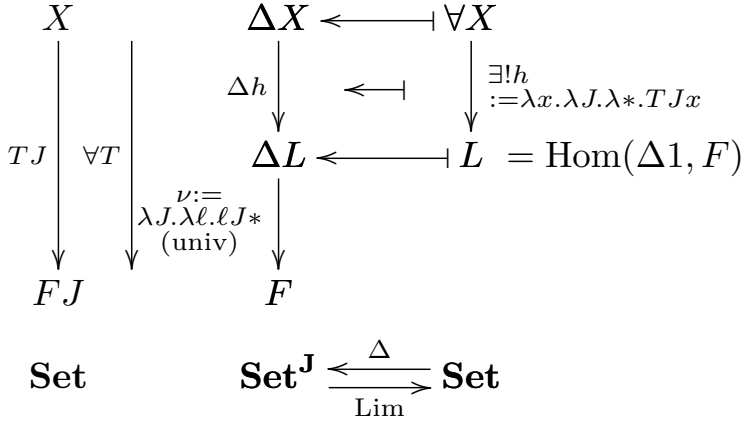
is a triple $(\ulcorner a \urcorner : 1 \to A, \ulcorner b \urcorner : 1 \to B, \ulcorner c \urcorner : 1 \to C)$,
but an element of $A \times_C B$ is a triple (a, b, c)...

(I'm omitting the "obeying certain properties"!)
(Why can I do that? Long story...)

Guessing ν: second attempt

...so let's try to guess ν, g, h by finding their types,
then finding natural constructions that yield
terms of those types, and then checking if
our guesses behave as expected...





$$
\begin{array}{rcl}
\ell & : & \Delta 1 \to F \\
\ell J & : & \Delta 1 J \to FJ \\
\ell J & : & 1 \to FJ \\
\ell J * & \in & FJ
\end{array}
\qquad
\begin{array}{rcl}
T & : & \Delta X \to F \\
TJ & : & \Delta X J \to FJ \\
TJ & : & X \to FJ \\
TJx & \in & FJ
\end{array}
$$

$$
\begin{array}{rcl}
\nu & : & \Delta L \to F \\
\nu J & : & \Delta L J \to FJ \\
\nu J & : & L \to FJ \\
\nu J \ell & \in & FJ
\end{array}
\qquad
\begin{array}{rcl}
h & : & X \to L \\
hx & \in & L \\
hx & : & \Delta 1 \to F \\
hxJ & : & \Delta 1 J \to FJ \\
hxJ & : & 1 \to FJ \\
hxJ* & \in & FJ
\end{array}
$$

$$
\begin{array}{rcl}
g & : & 1 \to L \\
g* & \in & L
\end{array}
$$

Let's try this:

$$
\begin{array}{rcll}
\nu J \ell & = & \ell J * & \Rightarrow \\
\nu J & = & \lambda \ell . \ell J * & \Rightarrow \\
\nu & := & \lambda J . \lambda \ell . \ell J * & \\
g* & = & \ell & \Rightarrow \\
g & := & \lambda * . \ell & \\
hxJ* & = & TJx & \Rightarrow \\
hxJ & = & \lambda * . TJx & \Rightarrow \\
hx & = & \lambda J . \lambda * . TJx & \Rightarrow \\
h & := & \lambda x . \lambda J . \lambda * . TJx &
\end{array}
$$

References

[BHL20] A. Bauer, P.G. Haselwarter, and P.L. Lumsdaine. "A General Definition of Dependent Type Theories". https://arxiv.org/pdf/2009.05539.pdf. 2020.

[BS07] J. Baez and M. Shulman. "Lectures on n-Categories and Cohomology". https://arxiv.org/pdf/math/0608420.pdf. 2007.

[CWM] S. Mac Lane. *Categories for the Working Mathematician (2nd ed.)* Springer, 1997.

[FavC] E. Ochs. "On my favorite conventions for drawing the missing diagrams in Category Theory". http:

//angg.twu.net/math-b.html#favorite-conventions. 2020.

[Gir10] J.-Y. Girard. *The Blind Spot*. European Mathematical Society, 2010.

[GLT89] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. http://www.paultaylor.eu/stable/prot.pdf. Cambridge, 1989.

[HOTT] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://saunders.phil.cmu.edu/book/hott-online.pdf. Institute for Advanced Study, 2013.

[Hut16] G. Hutton. *Programming in Haskell, 2nd ed.* Cambridge, 2016.

[IDARCT] E. Ochs. "Internal Diagrams and Archetypal Reasoning in Category Theory". In: *Logica Universalis* 7.3 (Sept. 2013). http://angg.twu.net/math-b.html#idarct, pp. 291–321.

[KLN04] F. Kamareddine, T. Laan, and R. Nederperlt. *A Modern Perspective on Type Theory.* Kluwer, 2004.

[Pie02] B. Pierce. *Types and Programming Languages.* MIT, 2002.

[Riehl] E. Riehl. *Category Theory in Context.* http://www.math.jhu.edu/~eriehl/context.pdf. Dover, 2016.

[Som00] G. Sommaruga. *History and Philosophy of Constructive Type Theory*. Springer, 2000.

[Zav20] V. Zavialov. "Haskell to Core: Understanding Haskell Features Through Their Desugaring". https://youtu.be/fty9QL4aSRc. 2020.