An introduction to Type Theory and to Pure Type Systems

and to the "Logic for Children" project

Eduardo Ochs (UFF, Brazil)
http://angg.twu.net/math-b.html#intro-tys-lfc

2019notes-types April 20, 2019 03:51

These slides are divided into...

Part 1: Discrete Mathematics
How I teach DM to our first-semester CompSci students.

Part 2: Logics, λ-Calculus and Translations
A course that some students attend on the second semester. It introduces the simply-typed λ-calculus, Curry-Howard, Planar Heyting Algebras, Intuitionistic Propositional Logic, Cartesian Closed Categories, the adjunction $(\times B) \dashv (B \to)$, and two functional languages: Lisp and Lua.

Part 3: Dependent types and Pure Type Systems

Eagle-eye view

In this set of slides we will learn how to interpret "judgments" and "pre-judgments" like these:

1) $a{:}\{1,2\}, b{:}\{2,3\}, a<b \vdash (10a+b){:}\mathbb{N}$
2) $a{:}\{1,2\}, b{:}\{2,3\} \vdash a<b$ $\Leftarrow$ a false proposition!
3) $A{:}\Theta, B{:}\Theta \vdash A{:}\Theta$
4) $A{:}\Theta, B{:}\Theta \vdash A \times B{:}\Theta$
5) $A{:}\Theta, B{:}\Theta, a{:}A, f{:}A \to B \vdash fa{:}B$
6) $A{:}\Theta, B{:}\Theta, a{:}A, b{:}B \vdash (a,b){:}A \times B$
7) $A{:}\Theta, B{:}\Theta, a{:}A \vdash (\lambda b{:}B.(a,b)){:}B \to A \times B$

in several type systems — some of them concrete but informal, some formal but very abstract...

A non-standard notion — "informally valid" — will help us understand which judgments are derivable.

Eagle-eye view (2)

We will see:

- The simply-typed λ-calculus ("$\lambda 1$") as a PTS
- Pure Type Systems in which we can represent:
  - Functors
  - Natural transformations
  - Truth-values, propositions, proofs

This is a preparation for:

- Interpreting categorical diagrams precisely
- Raising the status of CT diagrams from second-class to first-class citizens
- Formalizing the "Logic for Children" project

Motivation (for types)
The usual sales talk for type systems is:
Type systems are at the base of languages like Coq, that can be used to formalize a lot of mathematics, and in some cases they can fill up some details of the proofs themselves, automatically. Proof assistants have lots of uses in academia and in industry, which means jobs and grants...

But Coq is hard to learn =(, so:

Alternative motivation, for people with little free time:
Many programming languages use types and lambdas, even if in a primitive form; if we use types our objects become Lego-ish pieces that only match in a few ways...

Motivation (for types) (2)
At one point, when I was an undergrad, I took a course on Advanced Calculus that had a bit of Calculus of Variations... The characterization of "curves with minimal energy" only made sense to me when I discovered that I could draw it as:

$$\begin{tikzcd}
 & y & j \arrow[r] & L & \\
s \arrow[r] & {} \arrow[r] & {} & {} \arrow[r] & I \\
 & & x \arrow[uul] \arrow[uu] \arrow[uur] & &
\end{tikzcd}$$

I just had to name the types — and once I did that there was a single way to make the objects match and fit together! The construction became natural.

Motivation (for types) (3)

We will use Hindley/Seldin (2008):
"Lambda-Calculus and Combinators, an Introduction".
It presents several simple λ-calculi and type systems, but looks too abstract at first... we will see a way to understand some of its notations.

This book about Type Theory is mind-blowing and lots of fun: Kamareddine/Laan/Nederpelt (2004):
"A Modern Perspective on Type Theory —
From its origins until today"

We will use a book chapter by Thorsten Altenkirch, called "Naïve Type Theory", to complement the KLN book.
(`http://www.cs.nott.ac.uk/~psztxa/publ/fomus19.pdf`)

Part 1

Discrete Mathematics

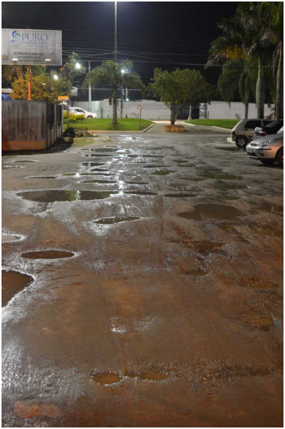
(At PURO/UFF)

(Let's start with) Discrete Mathematics
I teach in a city called Rio das Ostras, in the countryside of the state of Rio de Janeiro, in a big federal university ("UFF"), but in one of its smallest campi, away from the capital.
I sometimes teach Discrete Mathematics to Computer Science students there.

Many of the students there — even in CompSci — enter the university with very little knowledge of:
1) how to deal with variables,
2) how to write their calculations,
3) how to test their ideas.

Discrete Mathematics is a first-semester course there.
Let me explain my approach to fix (1), (2), and (3).

Pólo Universitário de Rio das Ostras (PURO)



The entrance of the
campus looks like this.
⇐ It feels like a nowhere.
I need to interact more
via the internets!!!

Discrete Mathematics at PURO/UFF
I structured the Discrete Mathematics ("DM") course in three layers.

Layer 1 consists of calculating things and learning how to use mathematical notation and definitions.
Layer 2 introduces some infinite objects, like $\mathbb{N}$ and $\mathbb{Z}$.
Layer 3 introduces a formal language for doing proofs.

Everything in Layer 1 can be calculated in a finite number of steps with very little creativity.

One of the basic things that we learn in Layer 1 is **set comprehensions** — that (ta-daaa! Magic!!!!!!) are the base for λ-calculus and Type Theory.

Basic Mathematical Objects

Here's a definition ("for adults") of the mathematical objects that we deal with in Level 1. Notation: $\mathcal{O}$ is the set of BMOs.

a) Numbers belong to $\mathcal{O}$; $\mathbb{Z} \subset \mathcal{O}$.
b) The truth-values belong to $\mathcal{O}$: $\{\mathbf{T}, \mathbf{F}\} \subset \mathcal{O}$.
c) $\mathcal{O}$ is closed by "finite lists" and "finite sets".
d) Finite strings belong to $\mathcal{O}$.

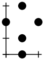
Item (d) is only introduced at the end of the course, when we show that "valid expressions" can be defined formally... "`1+2*(3+4)`" is "valid", but "`)+)`" is not.

Some graphical representations are allowed.

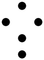
Some graphical representations

We define graphical representations for:
a) (finite) subsets of $\mathbb{Z}^2$,
b) functions whose domains are (a),
c) directed graphs...

$$K = \left\{ \begin{array}{ccc} & (1,3), & \\ (0,2), & & (2,2), \\ & (1,1), & \\ & (1,0) & \end{array} \right\} =$$

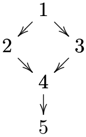


$=$



$$f = \left\{ \begin{array}{ccc} & ((1,3),1), & \\ ((0,2),0), & & ((2,2),2), \\ & ((1,1),1), & \\ & ((1,0),1) & \end{array} \right\} = \begin{array}{ccc} & 1 & \\ 0 & & 2 \\ & 1 & \\ & 1 & \end{array}$$

$$(V,A) = \left(\{1,2,3,4,5\}, \left\{ \begin{array}{c} (1,2),(1,3), \\ (2,4),(3,4), \\ (4,5) \end{array} \right\} \right) =$$

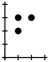


Layer 1: Set Comprehensions

One of the first things that I present to students is a syntax for set comprehensions using “generators”, “filters” and a “result expression”...

$$\{\underbrace{\underbrace{a \in \{1,2\}}_{\text{gen}}, \underbrace{b \in \{2,3\}}_{\text{gen}}, \underbrace{a < b}_{\text{filt}}}_{\text{context}};\ \underbrace{(a,b)}_{\text{expr}}\} \quad = \quad \{(1,2),(1,3),(2,3)\}$$

$$=$$



Note the ‘;’ instead of a ‘|’!

These things can be calculated from left to right using trees in a finite number of steps.

Layer 1: Set Comprehensions (2)

I make the students work in groups, and they solve the $5 + 19 + 16 + 9 + 16 + 7$ exercises quickly.

I suggest this table-ish way to draw the trees...

To calculate

$\{\, a \in \{1, 2\}, b \in \{2, 3\}, a < b \,;\; 10a + b \,\} = \{12, 13, 23\}$

we draw:

a	b	$a < b$	$10a + b$
1	2	**T**	12
1	3	**T**	13
2	2	**F**	$\mid$
2	3	**T**	23

the vertical bar means “abort”.

Valid values for the context: $(1, 2, \mathbf{T})$, $(1, 3, \mathbf{T})$, $(2, 3, \mathbf{T})$.

Layer 1: Set Comprehensions — SPOILER
Spoiler: the idea of context will be reused in many other contexts later, but with slightly different notations...
Compare:

1) $\{\, a \in \{1,2\}, b \in \{2,3\}, a < b \;;\; 10a+b \,\} = \{12, 13, 23\}$
2) $a{:}\{1,2\}, b{:}\{2,3\}, a < b \vdash (10a+b){:}\mathbb{N}$
3) $a{:}\{1,2\}, b{:}\{2,3\} \vdash a < b$ $\Leftarrow$ this is false!
4) $A{:}\mathsf{Sets}, B{:}\mathsf{Sets} \vdash A{:}\mathsf{Sets}$
5) $A{:}\mathsf{Sets}, B{:}\mathsf{Sets} \vdash A \times B{:}\mathsf{Sets}$
6) $A{:}\mathsf{Sets}, B{:}\mathsf{Sets}, a{:}A, f{:}A \to B \vdash f(a){:}B$
7) $A{:}\mathsf{Sets}, B{:}\mathsf{Sets}, a{:}A, b{:}B \vdash (a,b){:}A \times B$
8) $A{:}\mathsf{Sets}, B{:}\mathsf{Sets}, a{:}A \vdash (\lambda b{:}B.(a,b)){:}B \to A \times B$

An abuse of language
Remember that I make the students
do $5+19+16+9+16+7$ exercises about set comprehensions...
some of the exercises are about cartesian products, and
some introduce new notations, like:

$$\{x \in \{2,3,4\}, y \in \{2,3,4\}, y=3; (x,y)\}$$
$$\{x, y \in \{2,3,4\}, y=3; (x,y)\}$$
$$\{(x,y) \in \{2,3,4\}^2, y=3; (x,y)\}$$

This abuse of language will be incredibly important later
when we discuss type systems. We will allow things like:

$$A{:}\mathsf{Sets}, B{:}\mathsf{Sets}, (a,b){:}A \times B, f{:}A \to B \vdash fa{:}B$$

How do we make (a,b) behave as a variable?
(In the singular!)

An abuse of language (2)

Short answer:
(a, b) becomes a “long name” for a variable p,
a becomes an abbreviation for $\pi(a, b)$, i.e., πp,
b becomes an abbreviation for $\pi'(a, b)$, i.e., $\pi' p$,

$A{:}\mathsf{Sets}, B{:}\mathsf{Sets}, (a, b){:}A \times B, f{:}A \to B \vdash f(a){:}B$
becomes:
$A{:}\mathsf{Sets}, B{:}\mathsf{Sets}, p{:}A \times B, f{:}A \to B \vdash f(\pi p){:}B$

(More on “long names” later!)

λ-notation in Discrete Mathematics

In DM we see functions as sets of input-output pairs...

$$\text{If}\quad \begin{array}{rrcl} f: & \{1,2,3\} & \to & \{10,20,30\} \\ & x & \mapsto & 10x \end{array}$$

then $f = \left\{ \begin{array}{c} (1{,}10), \\ (2{,}20), \\ (3{,}30) \end{array} \right\}$.

The students learn that, e.g.,

$\left\{ \begin{array}{c} (1{,}10), \\ (2{,}20), \\ (3{,}30) \end{array} \right\}(2) = 20$, and $\left\{ \begin{array}{c} (1{,}10), \\ (2{,}20), \\ (3{,}30) \end{array} \right\}(4) =$ **ERROR**.

We see (in passing) that

$(\lambda x \in \{1,2,3\}.10x) = \{x \in \{1,2,3\}; (x, 10x)\} = \left\{ \begin{array}{c} (1{,}10), \\ (2{,}20), \\ (3{,}30) \end{array} \right\}$.

(Simultaneous) substitution
I also teach, right in the beginning of the course,
a notation for (simultaneous) substitution...
(Because I can't rely on the students' Portuguese!...)
Examples:

$$\begin{aligned} &((x+y)\cdot z)\begin{bmatrix} \scriptstyle x:=a+y \\ \scriptstyle y:=b+z \\ \scriptstyle z:=c+x \end{bmatrix} \\ = \quad &((a+y)+(b+z))\cdot(c+x) \end{aligned}$$

$$\begin{aligned} &(\text{Vanessão 20 reais})\Big[\utilde{\text{a} :}= \begin{pmatrix} \int \odot \\ \square \end{pmatrix}\Big] \\ = \quad &(\text{V}\begin{pmatrix} \int \odot \\ \square \end{pmatrix}\text{ness}\begin{pmatrix} \int \odot \\ \square \end{pmatrix}\text{o 20 re}\begin{pmatrix} \int \odot \\ \square \end{pmatrix}\text{is}) \end{aligned}$$

(Simultaneous) substitution (2)

This is useful to test equations,

$$
\begin{array}{l}
(x^2 - 4 = 0)\left[x := 3\right] = (9 - 4 = 0) = \mathbf{F} \\
(x^2 - 4 = 0)\left[x := 2\right] = (4 - 4 = 0) = \mathbf{T}
\end{array}
$$

and to define a way to calculate expressions with quantifiers:

$$
\begin{array}{lcl}
(\forall a \in \{2, 3, 5\}. a^2 < 10) & = & (a^2 < 10)[a := 2] \wedge \\
 & & (a^2 < 10)[a := 3] \wedge \\
 & & (a^2 < 10)[a := 5] \\
 & = & (2^2 < 10) \wedge (3^2 < 10) \wedge (4^2 < 10) \\
 & = & (4 < 10) \wedge (9 < 10) \wedge (16 < 10) \\
 & = & \mathbf{T} \wedge \mathbf{T} \wedge \mathbf{F} \\
 & = & \mathbf{F}
\end{array}
$$

Substitution and λ-calculus

...and I mention, very briefly, that we can use substitution to calculate with λ-terms:

$$\begin{aligned} (\lambda x \in \{1,2,3\}.10x)(2) &= (10x)[x := 2] \\ &= 10 \cdot 2 \\ &= 20 \end{aligned}$$

Some students later take an optional seminar course that is a very basic introduction to λ-calculus and Category Theory... In it they learn how to handle untyped λ-terms...

Part 2

λ-Calculus, Logics, and Translations

(At PURO/UFF)

An optional, second-semester-ish,
hands-on seminar course...
($\uparrow$ based on exercises)

I used it to test several ideas of the
“Logic for Children” project!...

Reductions

What happens if we forget all algebra that we know?
Suppose that we forget everything beyond basic arithmetic.
Now we only know how to add and multiply numbers.
What are the ways to calculate an expression —
$2 \cdot 3 + 4 \cdot 5$, say — step-by-step until we reach a final result?

$$\begin{tikzcd}
2 \cdot 3 + 4 \cdot 5 \arrow[r] \arrow[d] & 2 \cdot 3 + 20 \arrow[d] & \\
6 + 4 \cdot 5 \arrow[r] & 6 + 20 \arrow[r] & 26
\end{tikzcd}$$

We get a directed graph!

Reductions (2)

$$\begin{tikzcd}
2 \cdot 3 + 4 \cdot 5 \arrow[r] \arrow[d] & 2 \cdot 3 + 20 \arrow[d] & \\
6 + 4 \cdot 5 \arrow[r] & 6 + 20 \arrow[r] & 26
\end{tikzcd}$$

The students draw graphs like these for several initial expressions... each graph can be drawn in several equivalent ways, and the students learn to debug their graphs by working together, comparing their drawings, and trying to find more elegant shapes...

Reduction sequences

The arrows here show the one-step reductions starting from $2 \cdot 3 + 4 \cdot 5$.

$$\begin{tikzcd}
2 \cdot 3 + 4 \cdot 5 \arrow[r] \arrow[d] & 2 \cdot 3 + 20 \arrow[d] & \\
6 + 4 \cdot 5 \arrow[r] & 6 + 20 \arrow[r] & 26
\end{tikzcd}$$

After a bunch of similar exercices the students get the feeling that all reduction sequences terminate, and that we have “confluence”.

These properties — “termination” and “confluence” — are very important, and we want to keep them as we add more ways to calculate things...

Termination and confluence

From Hindley/Seldin (2008)...

P.14:

Note The property described in the Church-Rosser theorem, that if a term can be reduced to two different terms then these two terms can be further reduced to one term, is called *confluence*. The theorem states that *β-reduction is confluent*.

P.114:

The above theorem and its corollaries contrast strongly with untyped λ, in which reductions may be infinitely long and there is no decision-procedure for the relation $=_\beta$ (Corollary 5.6.3). They say that the world of typed terms is completely safe, in the sense that all computations terminate and their results are unique.

Untyped 'λ's

Typed 'λ's can be calculated using sets.
Untyped 'λ's have to be calculated using substituition.

$$\begin{array}{rcl}
(\lambda x \in \{1,2,3\}.10x)(2) & = & \left\{ \begin{array}{c} \text{(1,10),} \\ \text{(2,20),} \\ \text{(3,30)} \end{array} \right\} (2) \\
& = & 20
\end{array}$$

$$\begin{array}{rcl}
(\lambda x.10x)(2) & = & (10x)[x := 2] \\
& = & 10 \cdot 2 \\
& = & 20
\end{array}$$

Reducing functions and ‘λ’s

We add named functions and (untyped) ‘λ’s...

If $g(a) = a \cdot a + 4$ and $h = \lambda a.\, a \cdot a + 4$, then:

$$\begin{tikzcd}
g(2+3) \arrow[rr] \arrow[ddd] & & g(5) \arrow[ddddd] \\
& & \\
& & \\
(2+3)\cdot(2+3)+4 \arrow[dr] \arrow[dd] & & \\
& (2+3)\cdot 5+4 \arrow[dr] & \\
5\cdot(2+3)+4 \arrow[rr] & & 5\cdot 5+4 \arrow[d] \\
& & 25+4 \arrow[d] \\
& & 29
\end{tikzcd}$$

$$\begin{tikzcd}
h(2+3) \arrow[rr] \arrow[d] & & h(5) \arrow[d] \\
(\lambda a.\, a\cdot a+4)(2+3) \arrow[rr] \arrow[d] & & (\lambda a.\, a\cdot a+4)(5) \arrow[d] \\
(a\cdot a+4)[a:=2+3] \arrow[rr] \arrow[d] & & (a\cdot a+4)[a:=5] \arrow[ddd] \\
(2+3)\cdot(2+3)+4 \arrow[dr] \arrow[dd] & & \\
& (2+3)\cdot 5+4 \arrow[dr] & \\
5\cdot(2+3)+4 \arrow[rr] & & 5\cdot 5+4 \arrow[d] \\
& & 25+4 \arrow[d] \\
& & 29
\end{tikzcd}$$

$$\begin{tikzcd}
g(2+3) \arrow[rr] \arrow[d] & & g(5) \arrow[ddd] \\
(2+3)\cdot(2+3)+4 \arrow[dr] \arrow[dd] & & \\
& (2+3)\cdot 5+4 \arrow[dr] & \\
5\cdot(2+3)+4 \arrow[rr] & & 5\cdot 5+4 \arrow[d] \\
& & 25+4 \arrow[d] \\
& & 29
\end{tikzcd}$$

$$\begin{tikzcd}
h(2+3) \arrow[rr] \arrow[d] & & h(5) \arrow[d] \\
(\lambda a.\, a\cdot a+4)(2+3) \arrow[rr] \arrow[d] & & (\lambda a.\, a\cdot a+4)(5) \arrow[d] \\
(a\cdot a+4)[a := 2+3] \arrow[rr] \arrow[d] & & (a\cdot a+4)[a := 5] \arrow[ddd] \\
(2+3)\cdot(2+3)+4 \arrow[dr] \arrow[dd] & & \\
& (2+3)\cdot 5+4 \arrow[dr] & \\
5\cdot(2+3)+4 \arrow[rr] & & 5\cdot 5+4 \arrow[d] \\
& & 25+4 \arrow[d] \\
& & 29
\end{tikzcd}$$

Types

Let's (re)introduce types, but with another notation (':').

Let:
 $A = \{1,2\}$
 $B = \{30,40\}$.

If $f : A \to B$, then f is one of four functions...

$$f \in \left\{ \left\{ \begin{matrix} (1,30) \\ (2,30) \end{matrix} \right\}, \left\{ \begin{matrix} (1,30) \\ (2,40) \end{matrix} \right\}, \left\{ \begin{matrix} (1,40) \\ (2,30) \end{matrix} \right\}, \left\{ \begin{matrix} (1,40) \\ (2,40) \end{matrix} \right\} \right\}$$

Let's use the notation "$A \to B$" for
"the set of all functions from A to B".

Then $(A \to B) = \left\{ \left\{ \begin{matrix} (1,30) \\ (2,30) \end{matrix} \right\}, \left\{ \begin{matrix} (1,30) \\ (2,40) \end{matrix} \right\}, \left\{ \begin{matrix} (1,40) \\ (2,30) \end{matrix} \right\}, \left\{ \begin{matrix} (1,40) \\ (2,40) \end{matrix} \right\} \right\}$
and $f : A \to B$
means $f \in (A \to B)$.

Types (2)
In Type Theory and λ-calculus "$a : A$"
is pronounced "a is of type A", and the meaning
of this is *roughly* "$a \in B$".
(We'll see the differences between '$\in$' and ':' (much) later).

Note that:
1) if $f : A \to B$ and $a : A$ then $f(a) : B$
2) if $a : A$ and $b : B$ then $(a, b) : A \times B$
3) if $p : A \times B$ then $\pi p : A$ and $\pi' p : B$, where
4) 'π' means 'first projection' and
5) 'π'' means 'second projection'...

Example:
if $p = (2, 30)$ then $\pi p = 2$, $\pi' p = 30$.

Types (3)

Let:
$A = \{1, 2\}$,
$B = \{3, 4\}$,
$C = \{5, 6\}$.

If $p : A \times B$ and $g : B \to C$, then:

$$\underbrace{\underbrace{(\pi\ \underbrace{p}_{:A\times B}}_{:A},\quad \underbrace{\underbrace{g}_{:B\to C}\,(\underbrace{\pi'\ \underbrace{p}_{:A\times B}}_{:B})}_{:C})}_{:A\times C})$$

...because A, B, and C are known sets!

Types (4)

More abstractly now, if A, B, C are known sets, and $p : A \times B$ and $g : B \to C$, then:

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

i.e., $(\pi p, g(\pi' p)) : A \times C$.

Types (5)
At the right we see one of the standard notations for type inference:

$$\underbrace{(\underbrace{\pi\ \underbrace{p}_{:A\times B}}_{:A},\ \underbrace{\underbrace{g}_{:B\to C}\,(\underbrace{\pi'\ \underbrace{p}_{:A\times B}}_{:B}))}_{:C})}_{:A\times C}
\qquad
\dfrac{\dfrac{p{:}A\times B}{\pi p{:}A}\ \pi \qquad \dfrac{g{:}B\to C \qquad \dfrac{p{:}A\times B}{\pi' p{:}B}\ \pi'}{g(\pi' p){:}C}\ \mathsf{app}}{(\pi p, g(\pi' p)){:}A\times C}\ \mathsf{pair}$$

The tree at the right is a derivation of $(\pi p, g(\pi' p)){:}A\times C$ from hypotheses $p{:}A\times B$ and $g{:}B\to C$ (in a type system that is still unnamed)

Types (6)

We had these rules in the slide “Types (2)”...

1) if $f : A \to B$ and $a : A$ then $f(a) : B$
2) if $a : A$ and $b : B$ then $(a, b) : A \times B$
3) if $p : A \times B$ then $\pi p : A$ and $\pi' p : B$

They become tree rules, written with bars:

$$\dfrac{f{:}A{\to}B \quad a{:}A}{f(a)}\ \textsf{app}$$

$$\dfrac{a{:}A \quad b{:}B}{(a, b){:}A{\times}B}\ \textsf{pair} \qquad \dfrac{p{:}A{\times}B}{\pi p'{:}B}\ \pi' \qquad \dfrac{p{:}A{\times}B}{\pi p'{:}B}\ \pi'$$

Types (7)

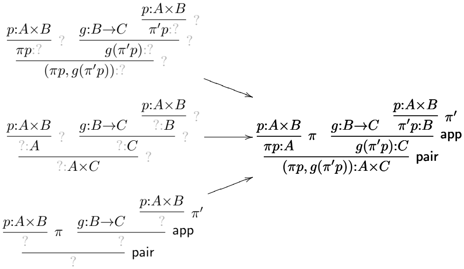
Each bar in a valid derivation is a substituition instance of one of the rules...

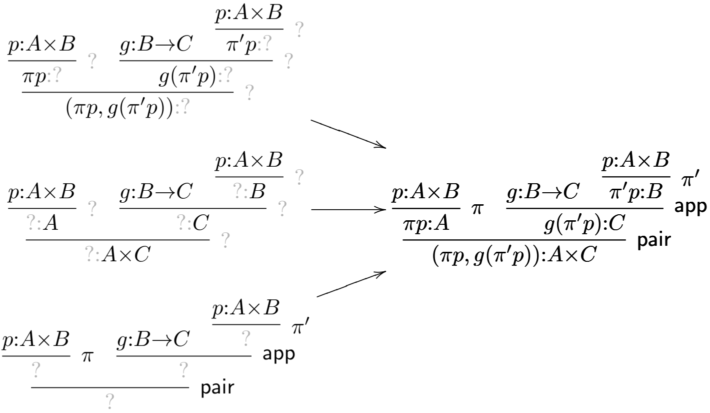
$$\dfrac{\dfrac{p{:}A{\times}B}{\pi p{:}A}\ \pi \qquad \dfrac{g{:}B{\to}C \qquad \dfrac{p{:}A{\times}B}{\pi' p{:}B}\ \pi'}{g(\pi' p){:}C}\ \mathsf{app}}{(\pi p, g(\pi' p)){:}A{\times}C}\ \mathsf{pair}$$

$$\left(\dfrac{a{:}A \quad b{:}B}{(a,b){:}A{\times}B}\ \mathsf{pair}\right)\left[\begin{smallmatrix} a:=\pi p \\ b:=g(\pi' p) \\ B:=C \end{smallmatrix}\right] = \left(\dfrac{\pi p{:}A \quad g(\pi' p){:}C}{(\pi p, g(\pi' p)){:}A{\times}C}\ \mathsf{pair}\right)$$

(Re)constructing missing information

Students are very good to learn syntax from examples.
Exercise: I ask the students to reconstruct the missing information in trees that have just a few term names, types, and rule names...





$$\dfrac{\dfrac{p{:}A\times B}{\pi p{:}?}\,? \qquad \dfrac{g{:}B\to C \qquad \dfrac{p{:}A\times B}{\pi' p{:}?}\,?}{g(\pi' p){:}?}\,?}{(\pi p, g(\pi' p)){:}?}\,?$$

$$\dfrac{\dfrac{p{:}A\times B}{?{:}A}\,? \qquad \dfrac{g{:}B\to C \qquad \dfrac{p{:}A\times B}{?{:}B}\,?}{?{:}C}\,?}{?{:}A\times C}\,?$$

$$\dfrac{\dfrac{p{:}A\times B}{?}\,\pi \qquad \dfrac{g{:}B\to C \qquad \dfrac{p{:}A\times B}{?}\,\pi'}{?}\,\mathsf{app}}{?}\,\mathsf{pair}$$

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

Erasing and reconstructing (and ‘λ’s)

Erasing information is easy.

Reconstructing information is harder
(and may not be always possible)...

What if the tree below is the result of taking
something bigger, and erasing information from it?

$$\dfrac{\dfrac{p{:}A\times B}{\pi p{:}A}\;\pi \qquad \dfrac{g{:}B\to C \quad \dfrac{p{:}A\times B}{\pi' p{:}B}\;\pi'}{g(\pi' p){:}C}\;\mathsf{app}}{(\pi p, g(\pi' p)){:}A\times C}\;\mathsf{pair}$$

This is the key idea for
understanding ‘λ’s and discharges!!!

Erasing and reconstructing (and ‘λ’s) (2)

What happens if we add to the left of each term,
or to the left of each “term : type”,
a list of free variables, written as “free variables $\vdash$”?

$$\dfrac{\dfrac{p}{\pi p} \qquad \dfrac{g \quad \dfrac{p}{\pi' p}}{g(\pi' p)}}{(\pi p, g(\pi' p))} \qquad \underset{\text{reconstruct}}{\overset{\text{erase}}{\overleftarrow{\dashrightarrow}}} \qquad \dfrac{\dfrac{p{:}A\times B}{\pi p{:}A}\,\pi \qquad \dfrac{g{:}B\to C \quad \dfrac{p{:}A\times B}{\pi' p{:}B}\,\pi'}{g(\pi' p){:}C}\,\mathsf{app}}{(\pi p, g(\pi' p)){:}A\times C}\,\mathsf{pair}$$

$$\dfrac{\dfrac{p \vdash p}{p \vdash \pi p} \qquad \dfrac{g \vdash g \quad \dfrac{p \vdash p}{p \vdash \pi' p}}{g, p \vdash g(\pi' p)}}{g, p \vdash (\pi p, g(\pi' p))} \qquad \underset{\text{reconstruct}}{\overset{\text{erase}}{\overleftarrow{\dashrightarrow}}} \qquad ?$$

Erasing and reconstructing (and 'λ's) (3)

In all the "standard" rules the context of the conclusion will be the union of the contexts of the hypotheses.

The new rule 'λ' is not standard.

It "discharges" a variable from the list of free variables, and transfers it to the right of the '$\vdash$', into the 'λ.

$$\dfrac{\dfrac{\dfrac{f \vdash f \quad a \vdash a}{f, a \vdash fa} \qquad \dfrac{g \vdash g \quad a \vdash a}{g, a \vdash ga}}{f, g, a \vdash (fa, ga)}}{f, g \vdash \lambda a.(fa, ga)}\ \lambda \qquad \dfrac{\dfrac{\dfrac{f{:}A\to B \quad a{:}A}{fa{:}B}\ \mathsf{app} \qquad \dfrac{g{:}A\to C \quad a{:}A}{ga{:}C}\ \mathsf{app}}{(fa, ga){:}B\times C}\ \mathsf{pair}}{(\lambda a{:}A.(fa, ga)){:}A\to(B\times C)}\ \lambda$$

How discharges are marked

Convention: the context is the list of undischarged hypotheses; a bar marked as "$\lambda; 1$" discharges the hypotheses marked "$[\cdot]^1$".

$$\dfrac{\dfrac{\dfrac{f \quad [a]^1}{fa} \qquad \dfrac{g \quad [a]^1}{ga}}{(fa, ga)}}{\lambda a.(fa, ga)}\ \lambda; 1 \qquad\qquad \dfrac{\dfrac{\dfrac{f \vdash f \quad a \vdash a}{f, a \vdash fa} \qquad \dfrac{g \vdash g \quad a \vdash a}{g, a \vdash ga}}{f, g, a \vdash (fa, ga)}}{f, g \vdash \lambda a.(fa, ga)}\ \lambda$$

The connection with Natural Deduction

Look at the tree below.
Each node of it is of the form “term : type”,
The contexts *(listing free variables)* are not shown,
the names of the rules are not shown (*except for ‘λ’)*,
but the discharges are marked...

$$\dfrac{\dfrac{\dfrac{f{:}A{\to}B \quad [a{:}A]^1}{fa{:}B} \qquad \dfrac{g{:}A{\to}C \quad [a{:}A]^1}{ga{:}C}}{(fa,ga){:}B{\times}C}}{(\lambda a{:}A.(fa,ga)){:}A{\to}(B{\times}C)}\ \lambda;1$$

What happens if we erase the terms and leave only the types?
And if after that we display the contexts?

The connection with Natural Deduction (2)

After erasing the terms we get this,

$$\dfrac{\dfrac{\dfrac{A\to B \quad [A]^1}{B} \qquad \dfrac{A\to C \quad [A]^1}{C}}{B\times C}}{A\to(B\times C)}\;1$$

After adding the contexts we get this:

$$\dfrac{\dfrac{\dfrac{A\to B\vdash A\to B \quad [A\vdash A]^1}{A\to B, A\vdash B} \qquad \dfrac{A\to C\vdash A\to C \quad [A\vdash A]^1}{A\to C, A\vdash C}}{A\to B, A\to C, A\vdash B\times C}}{A\to B, A\to C\vdash A\to(B\times C)}\;1$$

The connection with Natural Deduction (2)

If we change the notation a bit we get trees that talk about propositions and logic:

$$\dfrac{\dfrac{\dfrac{P\to Q \quad [P]^1}{Q} \qquad \dfrac{P\to R \quad [P]^1}{R}}{Q\land R}}{P\to(Q\land R)}\;1$$

$$\dfrac{\dfrac{\dfrac{P\to Q\vdash P\to Q \quad [P\vdash P]^1}{P\to Q, P\vdash Q} \qquad \dfrac{P\to R\vdash P\to R \quad [P\vdash P]^1}{P\to R, P\vdash R}}{P\to Q, P\to R, P\vdash Q\land R}}{P\to Q, P\to R\vdash P\to(Q\land R)}\;1$$

“If $P\to Q$ and $P\to R$ are true then $P\to(Q\land R)$ is true”.

Mixed judgments
In the beginning (on Discrete Maths) we saw contexts that mixed type declarations (working as generators) and propositions (working as filters)...

We looked at set comprehensions in detail:
$\{\, a \in \{1,2\}, b \in \{2,3\}, a < b \; ; \; 10a + b \,\} = \{12, 13, 23\}$

and I mentioned briefly "mixed" judgments like this one:
$a{:}\{1,2\}, b{:}\{2,3\}, a < b \vdash (10a + b){:}\mathbb{N}$

We can interpret these mixed judgments in λ-calculus if we use a trick called "propositions-as-types".

Propositions-as-types

We saw this tree ("in logic") as having only types:

$$\dfrac{\dfrac{\dfrac{P{\to}Q \quad [P]^1}{Q} \qquad \dfrac{P{\to}R \quad [P]^1}{R}}{Q{\land}R}}{P{\to}(Q{\land}R)}\;1$$

if we add "terms" to it we get:

$$\dfrac{\dfrac{\dfrac{f{:}P{\to}Q \quad [p{:}P]^1}{fp{:}Q} \qquad \dfrac{g{:}P{\to}R \quad [p{:}P]^1}{gp{:}R}}{(fp,gp){:}Q{\times}R}}{(\lambda p{:}P.(fp,gp)){:}P{\to}(Q{\times}R)}\;1$$

Propositions-as-types (2)

We can unify the notations by doing this:

$$\dfrac{\dfrac{\dfrac{f{:}\llbracket P{\to}Q\rrbracket \quad [p{:}\llbracket P\rrbracket]^1}{fp{:}\llbracket Q\rrbracket} \qquad \dfrac{[p{:}\llbracket P\rrbracket]^1 \quad g{:}\llbracket P{\to}R\rrbracket \quad [p{:}\llbracket P\rrbracket]^1}{gp{:}\llbracket R\rrbracket}}{(fp, gp){:}\llbracket Q{\wedge}R\rrbracket}}{(\lambda p{:}P.(fp, gp)){:}\llbracket P{\to}(Q{\wedge}R)\rrbracket}\ 1$$

where $\llbracket P{\to}Q\rrbracket := \llbracket P\rrbracket{\to}\llbracket Q\rrbracket$
and $\llbracket P{\wedge}Q\rrbracket := \llbracket P\rrbracket \times \llbracket Q\rrbracket$.

$\llbracket P\rrbracket$ is the “type of evidence” for the proposition P.

In the simplest model,
$\llbracket P\rrbracket = \emptyset$ when P is false, and
$\llbracket P\rrbracket$ is a singleton set when $\llbracket P\rrbracket$ is true.

Propositions-as-types (3)
Warning! Warning!

In the simplest model,
$[\![P]\!] = \emptyset$ when P is false, and
$[\![P]\!]$ is a singleton set when $[\![P]\!]$ is true.

However, adults usually prefer the "BHK interpretation",
in which $[\![P]\!]$ is the set of proofs for the proposition P...
See section 1.2.3 of Hermógenes Oliveira's PhD Dissertation:
"Inference Rules and the Meaning of the Logical Constants"
(2019).

Propositions-as-types (4)
A set is inhabited when it has an element.
A set $[\![P]\!]$ is inhabited iff P is true.
A proof of $P{\to}Q, P{\to}R \vdash P{\to}(Q{\land}R)$ must show that
if $[\![P{\to}Q]\!]$ and $[\![P{\to}R]\!]$ are inhabited
then $[\![P{\to}(Q{\land}R)]\!]$ is inhabited.
Here is a proof of $P{\to}Q, P{\to}R \vdash P{\to}(Q{\land}R)$:

$$f{:}[\![P{\to}Q]\!], g{:}[\![P{\to}R]\!] \vdash (\lambda p{:}P.(fp, gp)){:}[\![P{\to}(Q{\land}R)]\!]$$

This term $\lambda p{:}P.(fp, gp)$ can obtained by
1) finding a proof in ND for $P{\to}Q, P{\to}R \vdash P{\to}(Q{\land}R)$,
2) interpreting that proof as “types”,
3) reconstructing its “terms”...
i.e., by proof search followed by term inference.

The obvious term of type such-and-such
Suppose that we know a function $f : A \to B$ and a set C.
Then "f induces a function $(f{\times}C) : A{\times}C \to B{\times}C$
in a natural way".

How do we discover the function that
"deserves the name" $(f{\times}C)$?

Trick: "in a natural way" usually means
"using only the operations from λ-calculus", (!!!!!!!)
i.e., "a λ-term".

$$\frac{f{:}A{\to}B}{(f{\times}C){:}A{\times}C{\to}B{\times}C} \quad\Rightarrow\quad \frac{A{\to}B}{A{\times}C{\to}B{\times}C} \quad\Rightarrow (...)$$

$$\dfrac{A{\to}B}{\overline{A{\times}C{\to}B{\times}C}} \qquad \Rightarrow \qquad \dfrac{\dfrac{\dfrac{\dfrac{[A{\times}C]^1}{A} \qquad A{\to}B}{B} \qquad \dfrac{[A{\times}C]^1}{C}}{B{\times}C}}{A{\times}C{\to}B{\times}C}\;1$$

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

$$\Rightarrow \quad (f{\times}C) := (\lambda p{:}A{\times}C{:}(f(\pi p),\pi' p))$$

Cartesian Closed Categories

The ability to find
"the obvious term of type such-and-such"
is exactly what we need to learn Category Theory
starting by the "archetypal CCC", **Set**...

$$\begin{tikzcd}
A\times C \arrow[d, "(\times C)f"'] \arrow[dr, phantom, "\longmapsfrom"] & A \arrow[l, maps to] \arrow[d, "f"] \\
B\times C \arrow[d, "\substack{h^\flat \\ g}"'] \arrow[dr, phantom, "\substack{\longmapsfrom \\ \longmapsto}"] & B \arrow[l, maps to] \arrow[d, "\substack{h \\ g^\sharp}"] \\
D \arrow[r, maps to] \arrow[d, "k"'] \arrow[dr, phantom, "\longmapsto"] & C\to D \arrow[d, "(C\to)k"] \\
E \arrow[r, maps to] & C\to E
\end{tikzcd}$$

$$\begin{aligned}
(\times C)f &:= \lambda p{:}A\times C.(f(\pi p), \pi' p) \\
h^\flat &:= \lambda q{:}B\times C.(h(\pi q))(\pi' q) \\
g^\sharp &:= \lambda b{:}B.\lambda c{:}C.g(b,c) \\
(C\to)k &:= \lambda \varphi{:}C\to D.\lambda c{:}C.k(\varphi c)
\end{aligned}$$

Cartesian Closed Categories (2)

Fix an object C of **Set**. We have “obvious” functors $(\times C)$ and $(C{\to})$, and an adjunction $(\times C) \dashv (C{\to})$...
(“obvious” means “definable in λ-calculus!”)

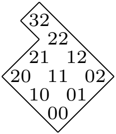
$$\begin{tikzcd}
A\times C \arrow[d, "(\times C)f"'] \arrow[dr, phantom, "\longmapsfrom"] & A \arrow[l, mapsto] \arrow[d, "f"] \\
B\times C \arrow[d, "\substack{h^\flat \\ g}"'] \arrow[dr, phantom, "\substack{\longmapsfrom \\ \longmapsto}"] & B \arrow[l, mapsto] \arrow[d, "\substack{h \\ g^\sharp}"] \\
D \arrow[r, mapsto] \arrow[d, "k"'] \arrow[dr, phantom, "\longmapsto"] & C\to D \arrow[d, "(C\to)k"] \\
E \arrow[r, mapsto] & C\to E
\end{tikzcd}$$

$$\begin{aligned}
(\times C)f &:= \lambda p{:}A\times C.(f(\pi p), \pi' p) \\
h^\flat &:= \lambda q{:}B\times C.(h(\pi q))(\pi' q) \\
g^\sharp &:= \lambda b{:}B.\lambda c{:}C.g(b, c) \\
(C\to)k &:= \lambda \varphi{:}C\to D.\lambda c{:}C.k(\varphi c)
\end{aligned}$$

“Logics”

Remember that the (second-semester) course is called “λ-Calculus, Logics, and Translations”...
The “Logics” is because the course also presents Planar Heyting Algebras, like this one:

$B =$



32
22
21 12
20 11 02
10 01
00

and we see that we can interpret Propositional Intuitionistic Logic (“PIL”) in B (and later in the course in any topology!), and that we have a “logical” adjunction $(\wedge Q) \dashv (Q{\to})$... but this doesn’t matter for these slides.

Part 3

Dependent types and Pure Type Systems

Notation:
$A{:}\Theta, B{:}\Theta, C{:}\Theta, p : A \times B, g : B \to C \vdash (\pi p, g(\pi' p)) : A \times C$

Judgments

Remember that in:
$\{\, a \in \{1,2\}, b \in \{2,3\}, a < b \;;\; 10a + b \,\} = \{12, 13, 23\}$
We also have a “context” at the left of the ‘$\vdash$’,
and we can read
$A{:}\Theta, B{:}\Theta, C{:}\Theta, p : A \times B, g : B \to C \vdash (\pi p, g(\pi' p)) : A \times C$
as: for every choice of
 a set A,
 a set B,
 a set C,
 $p : A \times B$,
 $g : B \to C$,
we can calculate $(\pi p, g(\pi' p))$ without errors,
and we will have $(\pi p, g(\pi' p)) \in A \times C$.

Judgments (2)

Remember that in

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

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"...

Judgments (3)

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”.

"Simple typing, Curry-style in λ"
(↑) This is the title of chapter 12 in Hindley/Seldin (2008). Here is a derivation in their system $\mathsf{TA}_{\lambda}^{\to}$, from p.161:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overset{1}{[x:\rho\to\sigma\to\tau]} \quad \overset{2}{[z:\rho]}}{xz:\sigma\to\tau}\,(\to \mathrm{e}) \quad \dfrac{\overset{3}{[y:\rho\to\sigma]} \quad \overset{2}{[z:\rho]}}{yz:\sigma}\,(\to \mathrm{e})}{xz(yz):\tau}\,(\to \mathrm{e})}{\lambda z.xz(yz):(\rho\to\tau)}\,(\to \mathrm{i}-2)}{\lambda yz.xz(yz):(\rho\to\sigma)\to\rho\to\tau}\,(\to \mathrm{i}-3)}{\lambda xyz.xz(yz):(\rho\to\sigma\to\tau)\to(\rho\to\sigma)\to\rho\to\tau}\,(\to \mathrm{i}-1)$$

“Simple typing, Curry-style in λ” (2)

If we take the previous derivation,

rename $\begin{bmatrix} \rho{:=}A \\ \sigma{:=}B \\ \tau{:=}C \\ z{:=}a \\ y{:=}f \\ x{:=}g \end{bmatrix}$ in it,

add some parentheses, and rewrite λgfa to $\lambda g.\lambda f.\lambda a$, we get:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overset{1}{[g : A \to (B \to C)]} \quad \overset{2}{[a : A]}}{ga : B \to C}\,(\to \mathrm{e}) \quad \dfrac{\overset{3}{[f : A \to B]} \quad \overset{2}{[a : A]}}{fa : B}\,(\to \mathrm{e})}{(ga)(fa) : C}\,(\to \mathrm{e})}{\lambda a.(ga)(fa) : A \to C}\,(\to \mathrm{i} - 2)}{\lambda f.\lambda a.(ga)(fa) : (A \to B) \to (A \to C)}\,(\to \mathrm{i} - 3)}{\lambda g.\lambda f.\lambda a.(ga)(fa) : (A \to (B \to C)) \to ((A \to B) \to (A \to C))}\,(\to \mathrm{i} - 1)$$

“Simple typing, Curry-style in λ” (3)

If we rename its rules to ‘app’ and ‘λ’
and change the notation for discharge of hypotheses a bit,
we get:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[g : A \to (B \to C)]^1 \quad [a : A]^2}{ga : B \to C}\ \mathsf{app} \qquad \dfrac{[f : A \to B]^3 \quad [a : A]^2}{fa : B}\ \mathsf{app}}{(ga)(fa) : C}\ \mathsf{app}}{\lambda a.(ga)(fa) : A \to C}\ \lambda ; 2}{\lambda f.\lambda a.(ga)(fa) : (A \to B) \to (A \to C)}\ \lambda ; 3}{\lambda g.\lambda f.\lambda a.(ga)(fa) : (A \to (B \to C)) \to ((A \to B) \to (A \to C))}\ \lambda ; 1$$

“Simple typing, Curry-style in λ” (4)

If we interpret a rule like ‘$\lambda;2$’ as
“introduce a ‘λ’ and discharge the hypothesis ‘2’ ”,
and we use a $\vdash$ to display the undischarged hypotheses
at some nodes starting from the bottom, we get:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[g{:}A\to(B\to C)]^1 \quad [a{:}A]^2}{ga{:}B\to C}\;\mathsf{app} \qquad \dfrac{[f{:}A\to B]^3 \quad [a{:}A]^2}{fa{:}B}\;\mathsf{app}}{a{:}A,\, f{:}A\to B,\, g{:}A\to(B\to C)\vdash(ga)(fa){:}C}\;\mathsf{app}}{f{:}A\to B,\, g{:}A\to(B\to C)\vdash\lambda a.(ga)(fa){:}A\to C}\;\lambda;2}{g{:}A\to(B\to C)\vdash\lambda f.\lambda a.(ga)(fa){:}(A\to B)\to(A\to C)}\;\lambda;3}{\vdash\lambda g.\lambda f.\lambda a.(ga)(fa){:}(A\to(B\to C))\to((A\to B)\to(A\to C))}\;\lambda;1$$

The undischarged hypotheses are exactly the free variables!

“Simple typing, Curry-style in λ” (5)

If we extend the idea

“The undischarged hypotheses are exactly the free variables”

to the nodes in the second line, inheriting the order from below — first a, then f, then g — we get:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[g{:}A\to(B\to C)]^1 \quad [a{:}A]^2}{a{:}A, g{:}A\to(B\to C)\vdash ga{:}B\to C}\;\mathsf{app} \qquad \dfrac{[f{:}A\to B]^3 \quad [a{:}A]^2}{a{:}A, f{:}A\to B\vdash fa{:}B}\;\mathsf{app}}{a{:}A, f{:}A\to B, g{:}A\to(B\to C)\vdash(ga)(fa){:}C}\;\mathsf{app}}{f{:}A\to B, g{:}A\to(B\to C)\vdash\lambda a.(ga)(fa){:}A\to C}\;\lambda;2}{g{:}A\to(B\to C)\vdash\lambda f.\lambda a.(ga)(fa){:}(A\to B)\to(A\to C)}\;\lambda;3}{\vdash\lambda g.\lambda f.\lambda a.(ga)(fa){:}(A\to(B\to C))\to((A\to B)\to(A\to C))}\;\lambda;1$$

“Simple typing, Curry-style in λ” (6)

The passage from the 2nd line to the 3rd line takes the contexts “a:..., g:... $\vdash$” and “a:..., f:... $\vdash$” above the line and produces a context “a:..., f:..., g:... $\vdash$” below, that is the union of the the contexts above:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[g{:}A\to(B\to C)]^1 \quad [a{:}A]^2}{a{:}A,\, g{:}A\to(B\to C) \vdash ga{:}B\to C}\;\mathsf{app} \qquad \dfrac{[f{:}A\to B]^3 \quad [a{:}A]^2}{a{:}A,\, f{:}A\to B \vdash fa{:}B}\;\mathsf{app}}{a{:}A,\, f{:}A\to B,\, g{:}A\to(B\to C) \vdash (ga)(fa){:}C}\;\mathsf{app}}{f{:}A\to B,\, g{:}A\to(B\to C) \vdash \lambda a.(ga)(fa){:}A\to C}\;\lambda;2}{g{:}A\to(B\to C) \vdash \lambda f.\lambda a.(ga)(fa){:}(A\to B)\to(A\to C)}\;\lambda;3}{\vdash \lambda g.\lambda f.\lambda a.(ga)(fa){:}(A\to(B\to C))\to((A\to B)\to(A\to C))}\;\lambda;1
$$

“Simple typing, Curry-style in λ” (7)

If we extend these two ideas — free variables and union — to the top line, we get this:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[g{:}A\to(B\to C)\vdash g{:}A\to(B\to C)]^1 \quad [a{:}A\vdash a{:}A]^2}{a{:}A, g{:}A\to(B\to C)\vdash ga{:}B\to C}\;\mathsf{app} \quad \dfrac{[f{:}A\to B\vdash f{:}A\to B]^3 \quad [a{:}A\vdash a{:}A]^2}{a{:}A, f{:}A\to B\vdash fa{:}B}\;\mathsf{app}}{a{:}A, f{:}A\to B, g{:}A\to(B\to C)\vdash (ga)(fa){:}C}\;\mathsf{app}}{f{:}A\to B, g{:}A\to(B\to C)\vdash \lambda a.(ga)(fa){:}A\to C}\;\lambda;2}{g{:}A\to(B\to C)\vdash \lambda f.\lambda a.(ga)(fa){:}(A\to B)\to(A\to C)}\;\lambda;3}{\vdash \lambda g.\lambda f.\lambda a.(ga)(fa){:}(A\to(B\to C))\to((A\to B)\to(A\to C))}\;\lambda;1$$

“Simple typing, Curry-style in λ” (8)

If we extend these two ideas to the top line
and we erase the things like ‘$[\ ]^2$’ and ‘$;2$’,
we get something whose intuitive semantics is very simple,
and that will be our motivation for more advanced
type systems...

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{g{:}A\to(B\to C)\vdash g{:}A\to(B\to C)} \qquad \dfrac{}{a{:}A\vdash a{:}A}}{a{:}A,\, g{:}A\to(B\to C)\vdash ga{:}B\to C}\,\mathsf{app} \qquad \dfrac{\dfrac{}{f{:}A\to B\vdash f{:}A\to B} \qquad \dfrac{}{a{:}A\vdash a{:}A}}{a{:}A,\, f{:}A\to B\vdash fa{:}B}\,\mathsf{app}}{a{:}A,\, f{:}A\to B,\, g{:}A\to(B\to C)\vdash (ga)(fa){:}C}\,\mathsf{app}}{f{:}A\to B,\, g{:}A\to(B\to C)\vdash \lambda a.(ga)(fa){:}A\to C}\,\lambda}{g{:}A\to(B\to C)\vdash \lambda f.\lambda a.(ga)(fa){:}(A\to B)\to(A\to C)}\,\lambda}{\vdash \lambda g.\lambda f.\lambda a.(ga)(fa){:}(A\to(B\to C))\to((A\to B)\to(A\to C))}\,\lambda}{}$$

...but this derivation suggests that the judgments in the top line are axioms. Let’s change this...

“Simple typing, Curry-style in λ” (9)

Let’s create a rule ‘v’ that works like this: “if $A{\to}B$ is a set then we can introduce a variable whose domain is $A{\to}B$”, and a rule ‘$\to$’ that works like this: “if A is a set and $B{\to}C$ is a set then $A{\to}(B{\to}C)$ is a set:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A{:}\Theta \quad \dfrac{B{:}\Theta \quad C{:}\Theta}{B{\to}C{:}\Theta}\;{\to}}{A{\to}(B{\to}C){:}\Theta}\;{\to}}{g{:}A{\to}(B{\to}C) \vdash g{:}A{\to}(B{\to}C)}\;\mathsf{v} \quad \dfrac{A{:}\Theta}{a{:}A \vdash a{:}A}\;\mathsf{v}}{a{:}A, g{:}A{\to}(B{\to}C) \vdash ga{:}B{\to}C}\;\mathsf{app} \quad \dfrac{\dfrac{\dfrac{A{:}\Theta \quad B{:}\Theta}{A{\to}B{:}\Theta}\;{\to}}{f{:}A{\to}B \vdash f{:}A{\to}B} \quad \dfrac{A{:}\Theta}{a{:}A \vdash a{:}A}}{a{:}A, f{:}A{\to}B \vdash fa{:}B}\;\mathsf{app}}{a{:}A, f{:}A{\to}B, g{:}A{\to}(B{\to}C) \vdash (ga)(fa){:}C}\;\mathsf{app}}{f{:}A{\to}B, g{:}A{\to}(B{\to}C) \vdash \lambda a.(ga)(fa){:}A{\to}C}\;\lambda}{g{:}A{\to}(B{\to}C) \vdash \lambda f.\lambda a.(ga)(fa){:}(A{\to}B){\to}(A{\to}C)}\;\lambda}{\vdash \lambda g.\lambda f.\lambda a.(ga)(fa){:}(A{\to}(B{\to}C)){\to}((A{\to}B){\to}(A{\to}C))}\;\lambda$$

“Simple typing, Curry-style in λ” (10)

If we fold, as in an accordion,
the middle part of the last derivation, we get:

$$\begin{array}{c} A{:}\Theta \quad B{:}\Theta \quad C{:}\Theta \\ \hline\hline \vdash \lambda g.\lambda f.\lambda a.(ga)(fa){:}(A{\to}(B{\to}C)){\to}((A{\to}B){\to}(A{\to}C)) \end{array}$$

and we can interpret this as:
if we know the values of A, B, C (and they are “:Θ”, i.e., sets),
then the judgement below the double bar is true.
For example, this holds when
$A = \{1, 2\}$, $B = \{3, 4\}$, $C = \{5, 6\}$.

Back to types

Remember that we had:

1) if $f : A \to B$ and $a : A$ then $f(a) : B$
2) if $a : A$ and $b : B$ then $(a, b) : A \times B$
3) if $p : A \times B$ then $\pi p : A$ and $\pi' p : B$

and:

$$\begin{array}{c} A{:}\Theta \quad B{:}\Theta \quad C{:}\Theta \\ \hline\hline \vdash \lambda g.\lambda f.\lambda a.(ga)(fa){:}(A{\to}(B{\to}C)){\to}((A{\to}B){\to}(A{\to}C)) \end{array}$$

back-to-types-2
Let's try to list all the rules that we saw up to this point.
We have rules that produce new sets, and
rules that produce new elements of sets.
The only rules that do something strange to the context
are the rule 'λ', that moves a hypothesis from the context
to the λ, and the rule 'v', the introduces a new variable
at both sides of the '$\vdash$'...

Impurities

We are using a non-standard trick here to present λ-calculus in a way that feels concrete. I call it "impurities".

I learned Pure Type Systems ages ago (2002?) from Herman Geuvers's PhD thesis. Everything was very abstract in it (no semantics! No intuition!), and very syntactical. He didn't explain what "pure" means...

Let me show an abstract of a talk that I gave in 2002.

“A System of Natural Deduction for Categories”
http://angg.twu.net/math-b.html#FMCS-2002
Abstract:

We will present a logic (system DNC) whose terms represent categories, objects, morphisms, functors, natural transformations, sets, points, and functions, and whose rules of deduction represent certain constructive operations involving those entities. Derivation trees in this system only represent the “T-part” (for “terms” and “types”) of the constructions, not the “P-part” (“proofs” and “propositions”): the rules that generate functors and natural transformations do not check that they obey the necessary equations. So, we can see derivations in this system either as constructions happening in a “syntactical world”, that should be related to the “real world” in some

way (maybe through meta-theorems that are yet to be found), or as being just "skeletons" of the real constructions, with the P-parts having been omitted for briefness.

(2nd paragraph deleted)

The way to formalize DNC, and to provide a translation between terms in its "logic" and the usual notations for Category Theory, is based on the following idea. Take a derivation tree D in the Calculus of Constructions, and erase all the contexts and all the typings that appear in it; also erase all the deduction steps that now look redundant. Call the new tree D'. If the original derivation, D, obeys some mild conditions, then it is possible to reconstruct it — modulo exchanges and unessential weakenings in the contexts — from D', that is much shorter. The algorithm that does the reconstruction

generates as a by-product a “dictionary” that tells the type and the “minimal context” for each term that appears in D'; by extending the language that the dictionary can deal with we get a way to translate DNC terms and trees — and also, curiously, with a few tricks more, and with some minimal information to “bootstrap” the dictionary, categorical diagrams written in a DNC-like language.
(End of the abstract.)

So
I started to add “impurities” to PTSs an in informal way, mixing PTS syntax and mathema
to be able to present judgments in a
I gave some talks that
To make a long story short, I added “impurities” to a Pure

Type System — in an informal way! — to be able to present PTS
Here is a part of the
I was working on a kind of “Curry-Howard for CT”
I gave some talks that year
I had to make some presentations in conferences that year
Note that we started from expressions with ‘λ’s and types that were quite concrete, like

Interlude: Logic for Children

From the webpage of the LfC workshop at the UniLog 2018:

The “children” in “logic for children” means “people without mathematical maturity”, which in its turn means people who:

- have trouble with very abstract definitions,

- prefer to start from particular cases (and then generalize),
- handle diagrams better than algebraic notations,
- like to use diagrams and analogies.

If we say that categorical definitions are "for adults" — because they may be very abstract — and that particular cases, diagrams, and analogies are "for children", then our intent with this workshop becomes easy to state. "Children" are willing to use "tools for children" to do mathematics, even if...

Logic for Children (2)
From the webpage of the LfC workshop at the UniLog 2018:

(...) "Children" are willing to use "tools for children" to do mathematics, even if they will have to translate everything to a language "for adults" to make their results dependable and publishable, and even if the bridge between their tools "for children" and "for adults" is somewhat defective, i.e., if the translation only works on simple cases...
We are interested in that bridge between maths "for adults" and "for children" in several areas. Maths "for children" are hard to publish, even informally as notes (see this thread in the Categories mailing list), so often techniques are rediscovered over and over, but kept restricted to the "oral culture" of the area.

Logic for Children: tools

We are going to use three “tools for children here”...

1. Parallel diagrams: particular / general

$$\begin{tikzcd}
\begin{pmatrix}\text{particular}\\ \text{case}\\ \text{“for children”}\end{pmatrix} \arrow[r, shift right, "\substack{\text{generalize}\\ \text{(hard)}}"'] & \begin{pmatrix}\text{general}\\ \text{case}\\ \text{“for adults”}\end{pmatrix} \arrow[l, shift right, "\substack{\text{particularize}\\ \text{(easy)}}"']
\end{tikzcd}$$

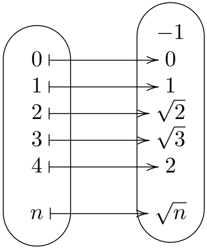
The diagrams for the general case and for a particular case have the same shape — or have very similar shapes.

Logic for Children: tools (2)

We are going to use three “tools for children here”...

2. External / internal views (‘→’ / ‘↦’)

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



$$\begin{tikzcd} \mathbb{N} \arrow[r, "\sqrt{}"] & \mathbb{R} \end{tikzcd}$$

Logic for Children: tools (3)

We are going to use three "tools for children here"...

3. The object "deserving a name"

$$\begin{tikzcd}
A \arrow[r, mapsto, "(\times C)"] \arrow[d, "f"'] & A{\times}C \arrow[d, "(\times C)f"] \\
B \arrow[r, mapsto] & B{\times}C
\arrow[from=1-1, to=2-2, phantom, "\mapsto"]
\end{tikzcd}
\qquad
\begin{tikzcd}
a \arrow[r, Rightarrow] \arrow[d, mapsto] & (a,c) \arrow[d, mapsto] \\
b \arrow[r, Rightarrow] & (b,c)
\arrow[from=1-1, to=2-2, phantom, "\mapsto"]
\end{tikzcd}$$

$$\begin{array}{rcl}
(\times C)(A) & := & A \times C \\
(\times C)(f) & := & ((a,c) \mapsto (b,c)) \\
 & = & (\lambda(a,c).(b,c)) \\
 & = & (\lambda(a,c).(f(a),c)) \\
 & = & (\lambda(a,c).(f(\pi(a,c)),\pi'(a,c))) \\
 & = & (\lambda p.(f(\pi p),\pi' p))
\end{array}$$

Note: I was obsessed by this idea for years, but it took me a long time to formalize it... it was incredibly useful as a private notation, but it seemed to work only in small examples. The trick to make it work is to use dictionaries.

Article: "Internal Diagrams and Archetypal Reasoning in Category Theory" [Ochs2013]. I still love about 80% of it, but I abandoned the idea of "downcasing types".

In the course about λ-Calculus, Logics, and Translation I don't even mention to the students that that paper exists.

End of the interlude

Let's go back to what our (2nd-semester) students see in the course!

Spaces of functions
Let: $A = \{1,2\}$, $B = \{3,4\}$, $C_1 = \{5,6\}$, $C_2 = \{7,8,9\}$.
If $f : A \to B$ then there are four possible values for f...
Trick: our first interpretation for ':' is '$\in$',
and $A \to B$ is B^A — the set of functions from A to B.

$f : A \to B$ means that f can be
$\{(1,3),(2,3)\}$ or
$\{(1,3),(2,4)\}$ or
$\{(1,4),(2,3)\}$ or
$\{(1,4),(2,4)\}$, so

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

Spaces of functions (2)
Let: $A = \{1, 2\}$, $B = \{3, 4\}$, $C_1 = \{5, 6\}$, $C_2 = \{7, 8, 9\}$.
If $f : A \to B$ then f takes each $a \in A$ to an element of B.
Let g be a function that takes each $a \in A$ to an element of C_a.

This means that g can be
$\{(1, 5), (2, 7)\}$ or
$\{(1, 5), (2, 8)\}$ or
$\{(1, 5), (2, 9)\}$ or
$\{(1, 6), (2, 7)\}$ or
$\{(1, 6), (2, 8)\}$ or
$\{(1, 6), (2, 9)\}$...

$f \in \{\, b_1 \in B, b_2 \in B;\ \{(1, b_1), (2, b_2)\}\,\} = \Pi a{:}A.B = (A \to A)$
$g \in \{\, c_1 \in C_1, c_2 \in C_2;\ \{(1, c_1), (2, c_2)\}\,\} = \Pi a{:}A.C_a$

Spaces of functions (3)

From last page:
$f \in \{\, b_1 \in B, b_2 \in B;\ \{(1,b_1),(2,b_2)\}\,\} = \Pi a{:}A.B = (A \to B)$
$g \in \{\, c_1 \in C_1, c_2 \in C_2;\ \{(1,c_1),(2,c_2)\}\,\} = \Pi a{:}A.C_a$

Compare:
$\{\, b_1 \in B, b_2 \in B;\ \{(1,b_1),(2,b_2)\}\,\} = \Pi a{:}\{1,2\}.B$
$\{\, c_1 \in C_1, c_2 \in C_2;\ \{(1,c_1),(2,c_2)\}\,\} = \Pi a{:}\{1,2\}.C_a$
$\{\, b_1 \in B, b_2 \in B;\ (b_1,b_2)\,\} = B \times B$
$\{\, c_1 \in C_1, c_2 \in C_2;\ (c_1,c_2)\,\} = C_1 \times C_2$

$\Pi i{:}\{7,42,99,200\}.D_i$ is similar to
$D_7 \times D_{42} \times D_{99} \times D_{200}$, but
the first returns functions and
the second return 4-uples.

Spaces of functions (4): dependent products
$\Pi i{:}\{7, 42, 99, 200\}.D_i$ is similar to
$D_7 \times D_{42} \times D_{99} \times D_{200}$, but
the first returns functions and
the second return 4-uples.

Terminology (half-weird):
$\Pi a{:}A.C_a$ and $\Pi i{:}I.D_i$ are dependent products.
Some type systems define $(A \to B) = B^A := \Pi a{:}A.B$.

Dependent products generaliize exponentials!!! Yuck!!!!

Pure Type Systems: first rules
Kamareddine/Laan/Nederperlt (2004), p.117:

$$\frac{}{\vdash s_1{:}s_2}\ ax \qquad\qquad (s_1,s_2)\in\mathbf{A}$$

$$\frac{\Gamma\vdash A{:}s}{\Gamma, x{:}A\vdash x{:}A}\ start$$

$$\frac{\Gamma\vdash A{:}B \quad \Gamma\vdash C{:}s}{\Gamma, x{:}C\vdash A{:}B}\ weak$$

$$\frac{\Gamma\vdash A{:}s_1 \quad \Gamma, x{:}A\vdash B{:}s_2}{\Gamma\vdash(\Pi x{:}A.B){:}s_3}\ \Pi \qquad\qquad (s_1,s_2,s_3)\in\mathbf{R}$$

$$\frac{\Gamma, a{:}A\vdash b{:}B \quad \Gamma\vdash(\Pi a{:}A.B){:}s}{\Gamma\vdash(\lambda a{:}A.b){:}(\Pi a{:}A.B)}\ \lambda$$

Specification: $(\mathbf{S},\mathbf{A},\mathbf{R})$ with $\mathbf{A}\subseteq\mathbf{S}^2$, $\mathbf{R}\subseteq\mathbf{S}^3$
For $\lambda 1$: $\mathbf{S}=\{*,\Box\}$, $\mathbf{A}=\{(*,\Box)\}$, $\mathbf{R}=\{(*,*,*)\}$

Pure Type Systems: first rules (2)
We will change the notation slightly, to:

$$\frac{}{\vdash s_1{:}s_2}\ a \qquad\qquad (s_1, s_2) \in \mathbf{A}$$

$$\frac{\Gamma \vdash A{:}s}{\Gamma, x{:}A \vdash x{:}A}\ v_s$$

$$\frac{\Gamma \vdash A{:}B \quad \Gamma \vdash C{:}s}{\Gamma, x{:}C \vdash A{:}B}\ w_s$$

$$\frac{\Gamma \vdash A{:}s_1 \quad \Gamma, x{:}A \vdash B{:}s_2}{\Gamma \vdash (\Pi x{:}A.B){:}s_3}\ \Pi_{s_1 s_2 s_3} \qquad (s_1, s_2, s_3) \in \mathbf{R}$$

$$\frac{\Gamma, a{:}A \vdash b{:}B \quad \Gamma \vdash (\Pi a{:}A.B){:}s}{\Gamma \vdash (\lambda a{:}A.b){:}(\Pi a{:}A.B)}\ \lambda$$

Specification: $(\mathbf{S}, \mathbf{A}, \mathbf{R})$ with $\mathbf{A} \subseteq \mathbf{S}^2$, $\mathbf{R} \subseteq \mathbf{S}^3$
For $\lambda 1$: $\mathbf{S} = \{\Theta, \Box\}$, $\mathbf{A} = \{(\Theta, \Box)\}$, $\mathbf{R} = \{(\Theta, \Theta, \Theta)\}$

Pure Type Systems: a derivation

Derivations in a PTS have *lots* of bureaucracy.
This is what we need to derive $A{:}\Theta, B{:}\Theta \vdash (\Pi a{:}A.B){:}\Theta$:

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

i.e.:

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

Pure Type Systems: a derivation (2)

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

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

Every variable is two levels below a sort

We had $A{:}\Theta{:}\Box$, $B{:}\Theta{:}\Box$ — A and B are sets,
and $a{:}A{:}\Theta$ — a is an element of a set.
We had $(A{\to}B) = (\Pi a{:}A.B)$, and $f{:}(\Pi a{:}A.B){:}\Theta$.
Convention on names of variables:
A, B (uppercase) for sets, a, b, f (lowercase) for elements.
The rules $v_\Box$ and $w_\Box$ introduce variables two levels below $\Box$.
the rules v_Θ and w_Θ introduce variables two levels below Θ.

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

More sorts

Some of the Pure Type Systems that we will see later will have specifications like $(\mathbf{S}, \mathbf{A}, \mathbf{R}) = (\{\Omega, \Theta, \Box\},\ \{(\Omega, \Box), (\Theta, \Box)\},\ ...)$. Their axioms are $\Omega{:}\Theta$ and $\Theta{:}\Box$ — i.e., they have $\Omega{:}\Theta{:}\Box$. Their variables are of four kinds:

$$\begin{array}{ccccccccc} & & \text{element} & : & \text{set} & : & \Theta & : & \Box \\ \text{witness} & : & \text{proposition} & : & \Omega & : & \Theta & & \end{array}$$

I prefer to think on that as:

$$\begin{array}{ccccccc} & & \text{element} & : & \text{set} & : & \Theta \\ \text{witness} & : & \text{proposition} & : & \Omega & & \end{array}$$

Convention on names of variables:

$$\begin{array}{ccccccc} & & a, b, f & : & A, B & : & \Theta \\ p, q & : & P, Q & : & \Omega & & \end{array}$$

PAT: Propositions As Types / Proofs As Terms

See Kamareddine/Laan/Nederperlt (2004), chapter 4, for PAT, Hermogenes Oliveira's PhD Thesis for the BHK interpretation. The standard way to think of

$$\begin{array}{ccccccc} & & \text{element} & : & \text{set} & : & \Theta \\ \text{witness} & : & \text{proposition} & : & \Omega & & \end{array}$$

is to interpret each proposition P as the set of its proofs, and a witness $p{:}P$ as an element of P; propositions that are false have no proofs.

I believe that we develop intuition about PAT quicker if we start with a model in which $\Omega = \{\mathbf{F}, \mathbf{T}\}$, $\mathbf{F}$ is an empty set, $\mathbf{T}$ is a singleton set; each proposition is interpreted as its truth-value, and all proofs/witnesses of a true proposition (i.e., elements of $\mathbf{T}$!) are identified.

Θ in the naïve model

[The most common error]

Lambda notation

Propositions and very small sets