Seminar notes about formalizing DNC using type systems
The version is the date at the footer line.

References/bibliography:

The slides for my presentation about DNC at the "Natural Deduction 2001" meeting:
`http://angg.twu.net/math/2001ndsl-{texsrc.tar,dvi.tar,ps}.gz`

Henk Barendregt, *Lambda Calculi with Types* (1992):
`http://citeseer.nj.nec.com/barendregt92lambda.html`
`ftp://ftp.cs.kun.nl/pub/CompMath.Found/HBKJ.ps.Z`

Izumi Takeuti, *The Theory of Parametricity in* λ*-cube* (2000?):
`http://www.sato.kuis.kyoto-u.ac.jp/~takeuti/art/cube.ps.gz`

Plotkin and Abadí, *A Logic for Parametric Polymorphism* (1993):
`http://citeseer.nj.nec.com/plotkin93logic.html`

Mark P. Jones, *First Class Polymorphism with Type Inference* (1997):
`http://citeseer.nj.nec.com/451312.html`
`http://www.cse.ogi.edu/~mpj/pubs/popl97-fcp.dvi.gz`

The list of references will grow a lot when I add the "dictionaries" — the tables showing how to translate the categorical notations found in some books and papers to DNC — to the notes.

Don't believe any idea here that you can't figure out the details yourself, or you've seen it stated in serious articles written by grown-ups. These notes contain unintentional errors and omissions. Feedback is not only welcome but also desperately needed. Etc, Etc.

Eduardo Ochs
`http://angg.twu.net/`
`http://www.mat.puc-rio.br/~edrx/`
`edrx@inx.com.br`

The standard notation for types in F

'$a : A$' means that the "value" of a belongs to A; however we are more interested in treating a as a term — a "program", that through a certain process of evaluation should become a value of A. Anyway, modulo evaluation, $a : A$ means $a \in A$. Note that A is a set, or something very close to that.

'$f : A \to B$' — same, but $A \to B$ is a convenient notation for B^A; $f \in B^A$. Note that $(\lambda a : A.b) : (A \to B)$ if $b : B$. The term b may contain occurrences of a.

'$f : \Pi A.B \to C$' — Here B and C may be type expressions, maybe involving occurrences of A. Example: $f : \Pi A.(A \to D) \to (A \to E)$. The symbol "$\Pi$", that in a sense works as a 'λ' for types, can be interpreted as a product. Suppose that there are only three types, A_1, A_2 and A_3, that can play the role of 'A'; then $f : \Pi A.(A \to D) \to (A \to E)$ — "f is a term that, for any choice of a type A, has type $B \to C$" — can be represented by a 3-uple $(f_{A_1}, f_{A_2}, f_{A_3})$, where $f_{A_i} : (A_i \to D) \to (A_i \to E)$; that is,

$$f \in \prod_{i=1,2,3} ((A_i \to D) \to (A_i \to E)).$$

In the general case we have

$$\Pi A.(A \to D) \to (A \to E) = \prod_{A \in *} (A \to D) \to (A \to E),$$

where the '*' stands for "the class of all possible sets A". This product is too large in the set-theoretical sense, which is why mathematicians[1] have such a hard time understanding polymorphic functions.

[1]i.e., me.

Terms in F

For example,

$$f^{\Pi A.(A\to D)\to(A\to E)} = \Lambda A.\lambda g^{A\to D}.\lambda a^{A}.h^{D\to E}\ (g\ a),$$

where $h^{D\to E}$ may be some known term of type $D \to E$, or just a variable (free or bound) of type $D \to E$; also D and E may be either type constants ortype variables, and in a certain sense (this is more clear if we think set-theoretically) $h^{D\to E}$ depends on D and E.

This f is a "function" that expects first a type to replace for the 'A's, then a $g^{A\to D}$, then an a^A; if we give these three things to f it will return an "element" of E.

It is common to use colons instead of superscripts and to make type annotations only at the 'λ's, so f can be written as:

$$(\Lambda A.\lambda g{:}(A \to D).\lambda a{:}A.h\ (g\ a)) : (\Pi A.(A \to D) \to (A \to E))$$

The type of h can then be inferred to be $D \to E$, but in some typing systems there is a formalized notion of a "contexts" that tells us that.

Keys to our notation

We'll be extending progressively the language used for the λ-cube ("λC"), up to the point where we will reach DNC^3, ("DNC-cube"), that will have λC and DNC as sublanguages.

'$\mathbf{E}$' means "the space of"; $\mathbf{E}_a$ is the space of 'a's: $a \in A$. All the rest of the world would usually write A for $\mathbf{E}_a$.

'$\mapsto$' is the "weak mapsto" symbol: an $a{\mapsto}b$ is a function that carries 'a's to 'b's — in general it won't be clear which b, and this is why our '$\mapsto$' is the *weak* mapsto; but it makes sense to speak of "an $a{\mapsto}b$": this will mean an arbitrary function from $\mathbf{E}_a$ to $\mathbf{E}_b$, that is, a point of $\mathbf{E}[a{\mapsto}b]$ where this is a space of functions, built from $\mathbf{E}_a$ and $\mathbf{E}_b$ in the obvious way.

In $a{\mapsto}b$ only the types (in a loose sense) are shown; we may add "term annotations" to them. For example, $(a_x{\mapsto}a_y{\mapsto}b_{f(x,y)})_f$ (i.e., $(a_x{\mapsto}(a_y{\mapsto}b_{f(x,y)}))_f$) is the function that expects a first 'a', sets x to this value, expects a second a, sets y to it, and returns a b, obtained as $f(x,y)$; the whole term, the function taking two parameters, is called f.

It is hoped that with '$\mapsto$' and '$\mathbf{E}$' we will in many cases be able to dispense with the explicit type annotations: '$a{\mapsto}b_{f(a)}{:}\mathbf{E}[a{\mapsto}b]$', for example, so $a{\mapsto}b_{f(a)}$ is already clear enough.

'$\mathbf{e}$' means "an element of": an $\mathbf{e}[A]$ is an element of A, and $\mathbf{e}[A]{:}A$.

The rules for the systems of the λ-cube (following Barendregt)
(Reference: `ftp://ftp.cs.kun.nl/pub/CompMath.Found/HBKJ.ps.Z`, p.89)

$$
\begin{array}{ll}
\text{(axiom)} & \dfrac{}{<>\vdash *{:}\Box} \\[2ex]
\text{(start-rule)} & \dfrac{\Gamma \vdash A{:}s}{\Gamma, x{:}A \vdash x{:}A}\ , x \notin \Gamma \\[2ex]
\text{(weakening rule)} & \dfrac{\Gamma \vdash A{:}B \quad \Gamma \vdash \mathbf{C}{:}s}{\Gamma, x{:}\mathbf{C} \vdash A{:}B}\ , x \notin \Gamma \\[2ex]
\text{(application rule)} & \dfrac{\Gamma \vdash F{:}(\Pi x{:}A.B) \quad \Gamma \vdash a{:}A}{\Gamma \vdash Fa{:}B[x{:}=a]} \\[2ex]
\text{(abstraction rule)} & \dfrac{\Gamma, x{:}A \vdash b{:}B \quad \Gamma \vdash (\Pi x{:}A.B){:}s}{\Gamma \vdash (\lambda x{:}A.b){:}(\Pi x{:}A.B)} \\[2ex]
\text{(conversion rule)} & \dfrac{\Gamma \vdash A{:}B \quad \Gamma \vdash B'{:}s \quad B =_\beta {}'' B'}{\Gamma \vdash A{:}B'} \\[2ex]
\text{(specific rules)} & \dfrac{\Gamma \vdash A{:}s_1 \quad \Gamma, x{:}A \vdash B{:}s_2}{\Gamma \vdash (\Pi x{:}A.B){:}s_2}
\end{array}
$$

x is a variable.
s, s_1, s_2 are sorts (i.e., $*$ or $\Box$).
All the systems of the λ-cube have the $(*.*)$ rule; the other three specific rules may be present or absent ($\to$ 8 different systems).

The λ-cube with explicit ranks, and with contexts ommited

(Oops, Barendregt uses the term "degree" instead of "rank"... see its p.110, and also the lemmas in pp.102ff).

Rank 3: $\Box$

Rank 2: $*, \alpha, \beta, \alpha', \beta', \ldots$ (kinds)

Rank 1: $A, B, A', B', F, \ldots$ (types, predicates)

Rank 0: $a, b, a', b', f, \ldots$ (terms, p-terms)

term : type : $*$: $\Box$

p-term : predicate : kind : $\Box$

"Types" are also called "formulas".

P-terms do not exist for kinds different from $*$.

When a symbol of a certain rank appears underlined in the rules it will mean that it must stand for a variable of that rank, not for an arbitrary expression of that rank.

$$\frac{}{*:}\ \text{ax}$$

$$\frac{\alpha{:}\Box}{\underline{A}{:}\alpha}\ \text{kstart} \qquad \frac{A{:}\alpha \quad \alpha'{:}\Box \quad \alpha =_\beta \alpha'}{A{:}\alpha'}\ \text{kconv}$$

$$\frac{A{:}*}{\underline{a}{:}A}\ \text{tstart} \qquad \frac{a{:}A \quad A'{:}* \quad A =_\beta A'}{a{:}A'}\ \text{tconv}$$

$$\frac{A{:}* \quad B{:}*}{(\Pi\underline{a}{:}A.B){:}*}\ (*,*) \qquad \frac{b{:}B \quad (\Pi\underline{a}{:}A.B){:}*}{(\lambda\underline{a}{:}A.b){:}(\Pi\underline{a}{:}A.B)}\ (*,*)\text{-abs} \qquad \frac{f{:}(\Pi\underline{a}{:}A.B) \quad a'{:}A}{fa'{:}B[\underline{a}{:}=a']}\ (*,*)\text{-app}$$

$$\frac{\alpha{:}\Box \quad B{:}*}{(\Pi\underline{A}{:}\alpha.B){:}*}\ (\Box,*) \qquad \frac{b{:}B \quad (\Pi\underline{A}{:}\alpha.B){:}*}{(\lambda\underline{A}{:}\alpha.b){:}(\Pi\underline{A}{:}\alpha.B)}\ (\Box,*)\text{-abs} \qquad \frac{f{:}(\Pi\underline{A}{:}\alpha.B) \quad A'{:}\alpha}{fA'{:}B[\underline{A}{:}=A']}\ (\Box,*)\text{-app}$$

$$\frac{A{:}* \quad \beta{:}\Box}{(\Pi\underline{a}{:}A.\beta){:}\Box}\ (*,\Box) \qquad \frac{B{:}\beta \quad (\Pi\underline{a}{:}A.\beta){:}}{(\lambda\underline{a}{:}A.B){:}(\Pi\underline{a}{:}A.\beta)}\ (*,\Box)\text{-abs} \qquad \frac{F{:}(\Pi\underline{a}{:}A.\beta) \quad a'{:}A}{Fa'{:}\beta[\underline{a}{:}=a']}\ (*,\Box)\text{-app}$$

$$\frac{\alpha{:}\Box \quad \beta{:}\Box}{(\Pi\underline{A}{:}\alpha.\beta){:}\Box}\ (\Box,\Box) \qquad \frac{B{:}\beta \quad (\Pi\underline{A}{:}\alpha.\beta){:}}{(\lambda\underline{A}{:}\alpha.B){:}(\Pi\underline{A}{:}\alpha.\beta)}\ (\Box,\Box)\text{-abs} \qquad \frac{F{:}(\Pi\underline{A}{:}\alpha.\beta) \quad A'{:}\alpha}{FA'{:}\beta[\underline{A}{:}=A']}\ (\Box,\Box)\text{-app}$$

Common abbreviations:

$v \to w := \Pi\underline{u}{:}v.w$ where $\underline{u}$ is "fresh", i.e., not in w

$\forall\underline{A}.v := \Pi\underline{A}{:}*.v$

$\Lambda\underline{A}.v := \lambda\underline{A}{:}*.v$

How some of these rules look in a DNC-ish notation

$$\begin{array}{rcl}
(\lambda a{:}A.b):(\Pi a{:}A.B){:}* & & a{\mapsto}b \;:\; \mathbf{E}[a{\mapsto}b] \;:* \\
(\lambda A{:}\alpha.b):(\Pi A{:}\alpha.B){:}* & & \mathbf{E}_a{\mapsto}b \;:\; \mathbf{E}[\mathbf{E}_a{\mapsto}b] \;:* \\
(\lambda a{:}A.B):(\Pi a{:}A.\beta):\Box & & a{\mapsto}\mathbf{E}_b \;:\; \mathbf{E}[a{\mapsto}\mathbf{E}_b] \;:\Box \\
(\lambda A{:}\alpha.B):(\Pi A{:}\alpha.\beta):\Box & & \mathbf{E}_a{\mapsto}\mathbf{E}_b{:}\mathbf{E}[\mathbf{E}_a{\mapsto}\mathbf{E}_b]{:}\Box
\end{array}$$

$$\frac{\mathbf{E}_a{:}* \quad \mathbf{E}_b{:}*}{\mathbf{E}[a{\mapsto}b]{:}*}\;(*,*) \qquad \frac{b{:}\mathbf{E}_b \quad \mathbf{E}[a{\mapsto}b]{:}*}{(a{\mapsto}b){:}\mathbf{E}[a{\mapsto}b]}\;(*,*)\text{-abs} \qquad \frac{f{:}\mathbf{E}[a{\mapsto}b] \quad a'{:}\mathbf{E}_a}{fa'{:}\mathbf{E}_b[\underline{a}{:}=a']}\;(*,*)\text{-app}$$

$$\frac{\mathbf{E}\mathbf{E}_a{:}\Box \quad \mathbf{E}_b{:}*}{\mathbf{E}[\mathbf{E}_a{\mapsto}b]{:}*}\;(\Box,*) \qquad \frac{b{:}\mathbf{E}_b \quad \mathbf{E}[\mathbf{E}_a{\mapsto}b]{:}*}{(\mathbf{E}_a{\mapsto}b){:}\mathbf{E}[\mathbf{E}_a{\mapsto}b]}\;(\Box,*)\text{-abs} \qquad \frac{f{:}\mathbf{E}[\mathbf{E}_a{\mapsto}b] \quad \mathbf{E}_{a'}{:}\mathbf{E}\mathbf{E}_a}{f\mathbf{E}_{a'}{:}\mathbf{E}_b[\mathbf{E}_a{:}=\mathbf{E}_{a'}]}\;(\Box,*)\text{-app}$$

$$\frac{\mathbf{E}_a{:}* \quad \mathbf{E}\mathbf{E}_b{:}\Box}{\mathbf{E}[a{\mapsto}\mathbf{E}_b]{:}\Box}\;(*,\Box) \qquad \frac{\mathbf{E}_b{:}\mathbf{E}\mathbf{E}_b \quad \mathbf{E}[a{\mapsto}\mathbf{E}_b]{:}\Box}{(a{\mapsto}\mathbf{E}_b){:}\mathbf{E}[a{\mapsto}\mathbf{E}_b]}\;(*,\Box)\text{-abs} \qquad \frac{F{:}\mathbf{E}[a{\mapsto}\mathbf{E}_b] \quad a'{:}\mathbf{E}_a}{Fa'{:}\mathbf{E}_b[\underline{a}{:}=a']}\;(*,\Box)\text{-app}$$

$$\frac{\mathbf{E}\mathbf{E}_a{:}\Box \quad \mathbf{E}\mathbf{E}_b{:}\Box}{\mathbf{E}[\mathbf{E}_a{\mapsto}\mathbf{E}_b]{:}\Box}\;(\Box,\Box) \qquad \frac{\mathbf{E}_b{:}\mathbf{E}\mathbf{E}_b \quad \mathbf{E}[\mathbf{E}_a{\mapsto}\mathbf{E}_b]{:}\Box}{(\mathbf{E}_a{\mapsto}\mathbf{E}_b){:}\mathbf{E}[\mathbf{E}_a{\mapsto}\mathbf{E}_b]}\;(\Box,\Box)\text{-abs} \qquad \frac{F{:}\mathbf{E}[\mathbf{E}_a{\mapsto}\mathbf{E}_b] \quad \mathbf{E}_{a'}{:}\mathbf{E}\mathbf{E}_a}{F\mathbf{E}_{a'}{:}\mathbf{E}\mathbf{E}_b[\mathbf{E}_a{:}=\mathbf{E}_{a'}]}\;(\Box,\Box)\text{-app}$$

The $(*,*)$-rules and the (usual) typed λ-calculus
Example:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{E}[a]{:}* \quad \mathbf{E}[b]{:}*}{\mathbf{E}[a{\mapsto}b]{:}*}\,(*,*)}{(a{\mapsto}b)_f{:}\mathbf{E}[a{\mapsto}b]}\,\text{tstart} \quad \dfrac{\mathbf{E}[a]{:}*}{a{:}\mathbf{E}[a]}\,\text{tstart}}{b_{f\,a}{:}\mathbf{E}[b]}\,(*,*)\text{-app} \quad \overline{\overline{\mathbf{E}[(a{\mapsto}b){\mapsto}b]{:}*}}}{((a{\mapsto}b)_f{\mapsto}b_{f\,a})_{\lambda f.f\,a}{:}\mathbf{E}[(a{\mapsto}b){\mapsto}b]}\,(*,*)\text{-abs}$$

If $\mathbf{E}_a$ and $\mathbf{E}_b$ are types (i.e., sets, in a sense, or domains) then we can form the type $\mathbf{E}[a{\mapsto}b]$, of functions from $\mathbf{E}_a$ to $\mathbf{E}_b$; and as $\mathbf{E}[a{\mapsto}b]$ is a type we can declare a variable of that type, f. In the same way we declare a variable $a{:}\mathbf{E}[a]$.

As the types of f and a match we can apply one to another, and obtain a term $f\,a$ (a b). The application rule that we need here is $(*,*)$-app; the trick to get the pair of sorts that appear in the name of an application is to take the "bigger type" involved, $\mathbf{E}[a{\mapsto}b]$ in this case, and check which (s_1, s_2)-rule would be used to form it; $(*,*)$, so it's $(*,*)$-app.

The last bar of the example is an abstraction. To pass from $f\,a$ to $\lambda f.f\,a$ we must check that the space where $\lambda f.f\,a$ would live would be admissible, i.e., if it would be a type or a kind; we derive $\mathbf{E}[(a{\mapsto}b){\mapsto}b]{:}*$, that is, it is a type, and so we are allowed to form $\lambda f.f\,a$; the f in the 'λ' could be any variable of the right type — the systems with contexts would say that in much more detail, but we have decided to be loose instead of formal and masochistic.

Again, the "$(*,*)$" in $(*,*)$-abs comes from the fact that the bigger "type" here, $E[(a{\mapsto}b){\mapsto}b]$, was constructed by a $(*,*)$ rule.

The $(\Box, *)$ rules and polymorphic terms

$$
\dfrac{
\dfrac{
\dfrac{
\mathbf{E}[\mathbf{E}_a]{:}\Box \qquad \dfrac{\mathbf{E}[a]{:}* \quad \mathbf{E}[a]{:}*}{\mathbf{E}[a{\mapsto}a]{:}*}\,(*,*)
}{
\mathbf{E}[\mathbf{E}_a{\mapsto}a{\mapsto}a]{:}*
}\,(\Box,*)
}{
(\mathbf{E}_a{\mapsto}a{\mapsto}a)_p{:}\mathbf{E}[\mathbf{E}_a{\mapsto}a{\mapsto}a]
}
\qquad
\dfrac{\mathbf{E}[b]{:}* \quad \mathbf{E}[c]{:}*}{\mathbf{E}[b{\mapsto}c]{:}*}\,(*,*)
}{
((b{\mapsto}c){\mapsto}(b{\mapsto}c))_{p\,\mathbf{E}[b\mapsto c]}{:}\mathbf{E}[(b{\mapsto}c){\mapsto}(b{\mapsto}c)]
}\,(\Box,*)\text{-app}
\qquad \cdots
}{
(\mathbf{E}_c{\mapsto}(b{\mapsto}c){\mapsto}(b{\mapsto}c))_{\lambda\mathbf{E}_c.p\,\mathbf{E}[b\mapsto c]}{:}\mathbf{E}[\mathbf{E}_c{\mapsto}(b{\mapsto}c){\mapsto}(b{\mapsto}c)]
}\,(\Box,*)\text{-abs}
$$

As $\mathbf{E}[\mathbf{E}_a]$ is a kind and $\mathbf{E}[a{\mapsto}a]$ is a term we can form, using the $(\Box, *)$ rule, the type $\mathbf{E}[\mathbf{E}_a{\mapsto}a{\mapsto}a]$, and we can declare a variable p (a polymorphic function) to be of that type; to apply something on p we need a $(\Box, *)$-app rule, and, after the substitutions, the result is a $(b{\mapsto}c){\mapsto}(b{\mapsto}c)$...

In the last bar we do a $(\Box, *)$-abstraction. We have $(p\,\mathbf{E}[b{\mapsto}c]){:}\mathbf{E}[(b{\mapsto}c){\mapsto}(b{\mapsto}c)]{:}*$, and the '…' stands for a derivation of $\mathbf{E}[\mathbf{E}_c{\mapsto}(b{\mapsto}c){\mapsto}(b{\mapsto}c)]{:}*$.

The $(*,\Box)$-rules: propositional logic

Notation: $\mathbf{E}[\mathrm{prf}P]$ is the space of "proofs that P is true".

A constructive proof that P implies Q is a function that, when given a proof of P, returns a proof of Q; that is, an arrow $\mathrm{prf}P{\mapsto}\mathrm{prf}Q$.

The truth-value of P is $\mathbf{E}[\mathrm{prf}P]$; intuitively the most important thing about that set is if it is empty (and thus P is false) or not (and then P will be true).

Note that we may force in a "model" that P is true, by creating the element "it is an axiom that P is true" and placing it in $\mathbf{E}[\mathrm{prf}P]$; intuitively that artifical proof of P should be as good as any other for using as a building block in bigger proofs; that will become clearer as we gain more experience with the system.

The truth-function of a predicate P defined over 'a's is a function that assigns, for each value of 'a', the truth-value $\mathbf{E}[\mathrm{prf}P(a)]$; each such $\mathbf{E}[\mathrm{prf}P(a)]$ will be an element of $*$, and so here is how we would type that:

$$\dfrac{\dfrac{\mathbf{E}[a]{:}* \qquad \dfrac{*:\Box}{\mathbf{E}[\mathbf{e}[*]]{:}\Box}\;\mathrm{ren}}{\mathbf{E}[a{\mapsto}\mathbf{e}[*]]{:}\Box}\;(*,\Box)}{(a{\mapsto}e[*]_{\mathbf{E}[\mathrm{prf}P(a)]})_P{:}\mathbf{E}[a{\mapsto}\mathbf{e}[*]]}\;\mathrm{kstart}
\qquad\qquad
\dfrac{\dfrac{A{:}* \qquad \dfrac{*:\Box}{*:\Box}}{(\Pi a{:}A.*){:}\Box}\;(*,\Box)}{P{:}(\Pi a{:}A.*)}\;\mathrm{kstart}$$

"$*{:}\Box$" says that the "set of types", $*$, is an element of the "set of kinds", $\Box$; so, an $\mathbf{e}[*]$ (an "element of $*$") is a type, and $\mathbf{E}[\mathbf{e}[*]]$ is the set of types again, an element of $\Box$. Using $(*,\Box)$ we discover that $\mathbf{E}[a{\mapsto}\mathbf{e}[*]]$ is a kind, and after that we declare a variable of that kind, P, and we make an annotation in its declaration saying that its $\mathbf{e}[*]$ is intended to mean $\mathbf{E}[\mathrm{prf}P(a)]$: we have $\mathbf{E}[\mathrm{prf}P(a)] = P\ a$.

A constructive proof of $\forall a.P(a)$ is a function that receives any 'a' from $\mathbf{E}_a$ and returns a $\mathrm{prf}P(a)$ in the corresponding $\mathbf{E}[\mathrm{prf}P(a)]$; that is, an arrow $a{\mapsto}\mathrm{prf}P(a)$. Such arrows are typed like this:

$$\dfrac{\dfrac{\dfrac{\dfrac{(a{\mapsto}\mathbf{e}[*]_{\mathbf{E}[\mathrm{prf}P(a)]})_P{:}\mathbf{E}[a{\mapsto}\mathbf{e}[*]] \qquad a{:}\mathbf{E}[a]}{(\mathbf{e}[*]_{\mathbf{E}[\mathrm{prf}P(a)]})_{P\,a}{:}\mathbf{E}[\mathbf{e}[*]]}\;(*,\Box)\text{-app}}{\mathbf{e}[*]_{\mathbf{E}[\mathrm{prf}P(a)]}{:}*}\;\mathrm{ren}}{\mathbf{E}[\mathrm{prf}P(a)]{:}*}\;\mathrm{ren}}{\mathrm{prf}P(a){:}\mathbf{E}[\mathrm{prf}P(a)]}\;\mathrm{tstart}
\qquad\qquad
\dfrac{\dfrac{\dfrac{\dfrac{P{:}(\Pi a{:}A.*) \qquad a{:}A}{P\,a{:}*}\;(*,)\text{-app}}{P\,a{:}*}}{P\,a{:}*}}{p{:}P\,a}\;\mathrm{tstart}$$

$$\dfrac{\dfrac{\mathrm{prf}P(a){:}\mathbf{E}[\mathrm{prf}P(a)] \qquad \dfrac{\mathbf{E}[a]{:}* \qquad \mathbf{E}[\mathrm{prf}P(a)]{:}*}{\mathbf{E}[a{\mapsto}\mathrm{prf}P(a)]{:}*}\;(*,*)}{a{\mapsto}\mathrm{prf}P(a){:}\mathbf{E}[a{\mapsto}\mathrm{prf}P(a)]}\;(*,*)\text{-abs}}{\mathrm{prf}\forall a.P(a){:}\mathbf{E}[\mathrm{prf}\forall a.P(a)]}\;\mathrm{ren}
\qquad\qquad
\dfrac{\dfrac{p{:}P\,a \qquad \dfrac{A{:}* \qquad P\,a{:}*}{(\Pi a{:}A.P\,a){:}*}\;(*,*)}{(\lambda a{:}A.p){:}(\Pi a{:}A.P\,a)}\;(*,*)\text{-abs}}{(\lambda a{:}A.p){:}(\Pi a{:}A.P\,a)}$$

And here is a proof of $(\forall x.\forall y.P(x,y))\supset(\forall x.P(x,x))$ using these ideas; we are supposing that $\mathbf{E}_x=\mathbf{E}_y$. Note that this tree is in "natural deduction" style, and its bars are instances of rules from a different set of rules — for example the abstractions here omit the right premiss, that was required before to check that the "type" of the conclusion was valid, and they may discharge a variable that appeared in a leaf above; I think that this discharge (of x, say) corresponds to removing a declaration $x{:}\mathbf{E}_x$ from the context, but I need to check that.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[(\mathrm{prf}\forall x.\forall y.P(x,y))_b]^2}{(x\mapsto y\mapsto \mathrm{prf}\,P(x,y))_b}\,\mathrm{ren} \qquad [x]^1}{(y\mapsto \mathrm{prf}\,P(x,y))_{b\,x}} \qquad [x]^1}{(\mathrm{prf}\,P(x,x))_{b\,x\,x}}}{(x\mapsto \mathrm{prf}\,P(x,x))_{\lambda x.b\,x\,x}}\,1}{(\mathrm{prf}\forall x.P(x,x))_{\lambda x.b\,x\,x}}\,\mathrm{ren}}{(\mathrm{prf}\forall x.\forall y.P(x,y))_b\mapsto(\mathrm{prf}\forall x.P(x,x))_{\lambda x.b\,x\,x}}\,2}{\mathrm{prf}\Big((\forall x.\forall y.P(x,y))\supset(\forall x.P(x,x))\Big)_{\lambda b.\lambda x.b\,x\,x}}\,\mathrm{ren}
$$

To make things a bit more formal let's translate the tree above to one in the notation of the λ-cube with explicit ranks, without contexts; the rules will still be the ones in natural deduction style, that we haven't stated precisely yet. P is declared as $P{:}(\Pi x{:}X.\Pi y{:}X.*){:}\Box$.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[b{:}(\Pi x{:}X.\Pi y{:}X.Pxy)]^2}{b{:}(\Pi x{:}X.\Pi y{:}X.Pxy)} \qquad [x{:}X]^1}{bx{:}(\Pi y{:}X.Pxy)} \qquad [x{:}X]^1}{bxx{:}Pxx}}{(\lambda x{:}X.bxx){:}(\Pi x{:}X.Pxx)}\,1}{(\lambda x{:}X.bxx){:}(\Pi x{:}X.Pxx)}}{(\lambda b{:}(\Pi x{:}X.\Pi y{:}X.Pxy).\lambda x{:}X.bxx){:}(\Pi b{:}(\Pi x{:}X.\Pi y{:}X.Pxy).\Pi x{:}X.Pxx)}\,2}{(\lambda b{:}(\Pi x{:}X.\Pi y{:}X.Pxy).\lambda x{:}X.bxx){:}(\Pi b{:}(\Pi x{:}X.\Pi y{:}X.Pxy).\Pi x{:}X.Pxx)}
$$

Note: this example was taken from p.87 of [Barendregt].

Using the $(\Box,*)$- and $(\Box,\Box)$-rules it will be possible to quantify over all the 'P's of the right kind; we will see that later.

Representing categories in the λ-cube

A category $\mathbf{C}$ is represented as a structure (in the metalanguage) consisting of a kind $\mathbf{E}[\mathbf{o}[\mathbf{C}]]$, the "space of objects of $\mathbf{C}$", a type $\mathrm{Hom}_{\mathbf{C}}$, that gives for any two objects A and B of $\mathbf{C}$ the set (i.e., the type) of morphisms going from A to B, terms $\mathrm{id}_{\mathbf{C}}$ and $\circ_{\mathbf{C}}$ with the obvious meaning, a type $\mathrm{Eqm}_{\mathbf{C}}$, that for any two morphisms in the same hom-set answers the set of proof that they are equal, and seven "prf"-like terms that state that things behave as they should.

Typings, conventions, and abbreviations (":=" means "is defined as"):

$\mathbf{EoC}{:}\Box$
$A, B, C, D{:}\mathbf{EoC}$

$\mathrm{Hom}_{\mathbf{C}}{:}\mathbf{E}[\mathbf{oC}{\mapsto}\mathbf{oC}{\mapsto}\mathbf{e}[*]]$
$(\mathbf{oC}_A{\mapsto}\mathbf{oC}_B{\mapsto}\mathbf{e}[*]_{\mathrm{Hom}_{\mathbf{C}}(A,B)})_{\mathrm{Hom}_{\mathbf{C}}}$
$f : A \to B \qquad := \qquad f{:}(\mathrm{Hom}_{\mathbf{C}}\, A\, B),$
$f, f' : A \to B$
$g, g' : B \to C$
$h, h' : C \to D$

$\mathrm{id}_{\mathbf{C}}{:}\mathbf{E}[\mathbf{oC}_A{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{C}}\, A\, A]]$
$(\mathbf{oC}_A{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{C}}\, A\, A]_{\mathrm{id}_A})_{\mathrm{id}_{\mathbf{C}}}$

$\circ_{\mathbf{C}}{:}\mathbf{E}[\mathbf{oC}_A{\mapsto}\mathbf{oC}_B{\mapsto}\mathbf{oC}_B{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{C}}\, B\, C]{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{C}}\, A\, B]{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{C}}\, A\, C]]$
$(\mathbf{oC}_A{\mapsto}\mathbf{oC}_B{\mapsto}\mathbf{oC}_B{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{C}}\, B\, C]_g{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{C}}\, A\, B]_f{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{C}}\, A\, C]_{g\circ f})_{\circ_{\mathbf{C}}}$
$g \circ f \qquad := \qquad \circ_{\mathbf{C}}\, A\, B\, C\, g\, f$

$\mathrm{Eqm}_{\mathbf{C}}{:}\mathbf{E}[\mathbf{oC}_A{\mapsto}\mathbf{oC}_B{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{C}}\, A\, B]_f{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{C}}\, A\, B]_{f'}{\mapsto}\mathbf{e}[*]_{\mathbf{E}\mathrm{prf}[f=f']}$
$(\mathbf{oC}_A{\mapsto}\mathbf{oC}_B{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{C}}\, A\, B]_f{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{C}}\, A\, B]_{f'}{\mapsto}\mathbf{e}[*]_{\mathbf{E}\mathrm{prf}[f=f']})_{\mathrm{Eqm}_{\mathbf{C}}}$
$\mathbf{E}\mathrm{prf}[f = f'] \qquad := \qquad \mathrm{Eqm}_{\mathbf{C}}\, A\, B\, f\, f'$
$\mathrm{prf}[f = f']{:}\mathbf{E}\mathrm{prf}[f = f']$

The seven "prf"-like terms:

$\mathrm{prf}[\forall A.\forall B.\forall f : A \to B. f = f]$
$\mathrm{prf}[\forall A.\forall B.\forall f, f' : A \to B.(f = f' \supset f' = f)]$
$\mathrm{prf}[\forall A.\forall B.\forall f, f', f'' : A \to B.(f = f' \supset (f' = f'' \supset f = f''))]$

$\mathrm{prf}[\forall A.\forall B.\forall C.\forall f, f' : A \to B.\forall g, g' : B \to C.$
$\qquad (f = f' \supset (g = g' \supset g \circ f = g' \circ f'))]$

$\mathrm{prf}[\forall A.\forall B.\forall f : A \to B.(\mathrm{id}_B \circ f = f)]$
$\mathrm{prf}[\forall A.\forall B.\forall f : A \to B.(f \circ \mathrm{id}_A = f)]$

Representing functors and natural transformations

A functor F from $\mathbf{C}$ to $\mathbf{D}$ is represented by a kind for its action on objects,
$F_o{:}\mathbf{E}[\mathbf{oC}{\mapsto}\mathbf{oD}]$
$(\mathbf{oC}_A{\mapsto}\mathbf{oD}_{F(A)})_{F_m}$
$F(A) \quad := \quad F_o\,A,$
a term for its action on morphisms,
$(\mathbf{oC}_A{\mapsto}\mathbf{oC}_B{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{C}}\,A\,B]_f{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{C}}\,(F\,A)\,(F\,B)]_{F(f)})_{F_m}$
$F(f) \quad := \quad F_m\,A\,B\,f,$
terms that assert that F respects identities and composition,
$\mathrm{prf}[\forall A.F(\mathrm{id}_A) = \mathrm{id}_{F(A)}]$
$\mathrm{prf}[\forall A.\forall B.\forall C.\forall f : A \to B.\forall g : B \to C.F(g \circ f) = F(g) \circ F(f)],$
and a term that says that equal terms are carried into equal terms:
$\mathrm{prf}[\forall A.\forall B.\forall f, f' : A \to B.(f = f' \supset F(f) = F(f'))]$

A natural transformation T from F to G, where F and G are functors from $\mathbf{C}$ to $\mathbf{D}$, is represented by a term
$(\mathbf{oC}_A{\mapsto}\mathbf{e}[\mathrm{Hom}_{\mathbf{D}}\,(F\,A)\,(G\,A)]_{T(A)})_T$
and a proof that it respects the square condition:
$\mathrm{prf}[\forall A.\forall B.\forall f : A \to B.(T(B) \circ F(f) = G(f) \circ T(A))]$

Homework for me: search the literature for systems that “have equality”, and where the equality behaves well automatically. Maybe by having a polymorphic term ‘=’ and a ‘$=_A$’ for each type, with polymorphic terms that assert that each ‘$=_A$’ is a transitive and reflexive relation, and that equal functions carry equal values to equal results... and then all my other assertions about specific equalities will refer to these ‘$=_A$’s, that are already transitive and reflexive...

The articles of Plotkin and Abadí and of Takeuti Izumi use systems with equality, but I don’t understand them very well...

The $(\Box, *)$-rules again: $\land, \lor, \bot, \exists$
Fact: we can define $\mathrm{prf}[P \land Q], \mathrm{prf}[P \lor Q], \mathrm{prf}[\bot]$ and $\mathrm{prf}[\exists x.P(x)]$ by:

$\mathbf{E}[\mathrm{prf}[P \land Q]] := \mathbf{E}[\mathbf{E}_a \mapsto (\mathrm{prf}P \mapsto \mathrm{prf}Q \mapsto a) \mapsto a]$
$\mathbf{E}[\mathrm{prf}[P \lor Q]] := \mathbf{E}[\mathbf{E}_a \mapsto (\mathrm{prf}P \mapsto a) \mapsto (\mathrm{prf}Q \mapsto a) \mapsto a]$
$\mathbf{E}[\bot] := \mathbf{E}[\mathbf{E}_a \mapsto a]$
$\mathbf{E}[\exists x.P(x)] := \mathbf{E}[\mathbf{E}_a \mapsto (x \mapsto \mathrm{prf}P(x) \mapsto a) \mapsto a]$

and the "logical rules" below all become interpretable as standard operations om λC-terms (and even as combinators):

$$\frac{\mathrm{prf}[P \land Q]}{\mathrm{prf}P} \qquad \frac{\mathrm{prf}[P \land Q]}{\mathrm{prf}Q} \qquad \frac{\mathrm{prf}P \quad \mathrm{prf}Q}{\mathrm{prf}[P \land Q]}$$

$$\frac{\mathrm{prf}P}{\mathrm{prf}[P \lor Q]} \qquad \frac{\mathrm{prf}Q}{\mathrm{prf}[P \lor Q]} \qquad \frac{\mathrm{prf}[P \lor Q] \quad \mathrm{prf}P \mapsto \mathrm{prf}A \quad \mathrm{prf}Q \mapsto \mathrm{prf}A}{\mathrm{prf}A}$$

$$\frac{\mathrm{prf}[\bot]}{\mathrm{prf}A}$$

$$\frac{x \quad \mathrm{prf}P(x)}{\mathrm{prf}[\exists x.P(x)]} \qquad \frac{\mathrm{prf}[\exists x.P(x)] \quad x \mapsto \mathrm{prf}P(x) \mapsto \mathrm{prf}A}{\mathrm{prf}A}$$

(the rules for $\exists$ will probably look very obscure now).

Two examples: $\land$E (in functional notation, without contexts),

$$\frac{a, b}{a} \qquad \dfrac{\dfrac{\dfrac{[a]^2}{b \mapsto a}\,1}{a \mapsto b \mapsto a}\,2 \quad \dfrac{\mathbf{E}_a \quad \mathbf{E}_x \mapsto (a \mapsto b \mapsto x) \mapsto x}{(a \mapsto b \mapsto a) \mapsto a}}{a}$$

$$\dfrac{\dfrac{\dfrac{a{:}A}{(\lambda b{:}B.a){:}(\Pi b{:}B.A)}}{(\lambda a{:}A.\lambda b{:}B.a){:}(\Pi a{:}A.\Pi b{:}B.A)} \quad \dfrac{A{:}* \quad p{:}(\Pi X{:}*.\Pi f{:}(\Pi a{:}A.\Pi b{:}B.X).X)}{pA{:}(\Pi f{:}(\Pi a{:}A.\Pi b{:}B.A).A)}}{pA(\lambda a{:}A.\lambda b{:}B.a){:}A}$$

and $\land$I:

$$\frac{a \quad b}{a, b} \qquad \dfrac{\dfrac{\dfrac{b \quad \dfrac{a \quad [a \mapsto b \mapsto x]^1}{b \mapsto x}}{x}}{(a \mapsto b \mapsto x) \mapsto x}\,1}{\mathbf{E}_x \mapsto (a \mapsto b \mapsto x) \mapsto x}\,2$$

The two flavours of natural deduction

$$\dfrac{\dfrac{\dfrac{[a]^1 \quad b}{a, b} \qquad a, b{\mapsto}c}{c}}{a{\mapsto}b}\;1 \qquad\qquad \dfrac{\dfrac{\dfrac{[\mathrm{prf}P]^1 \quad \mathrm{prf}Q}{\mathrm{prf}[P \wedge Q]} \qquad \mathrm{prf}[P \wedge Q{\to}R]}{\mathrm{prf}[R]}}{\mathrm{prf}[P{\to}R]}\;1$$

(Functional/logical).
They meet in the rules for the quantifiers.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x]^3 \quad [Pxy]^1}{\exists x.Pxy}\;\exists\mathrm{I} \qquad \dfrac{[y]^2 \quad \forall y.((\exists x.Pxy){\to}Qy)}{(\exists x.Pxy){\to}Qy}\;\forall\mathbf{E}}{Qy}}{Pxy{\to}Qy}\;1}{\forall y.(Pxy{\to}Qy)}\;\forall\mathrm{I};2}{\forall x.\forall y.(Pxy{\to}Qy)}\;\forall\mathrm{I};3$$

$$\dfrac{\dfrac{\dfrac{[\exists x.Pxy]^2 \qquad \dfrac{[Pxy]^1 \qquad \dfrac{[y]^3 \qquad \dfrac{[x]^1 \quad \forall x.\forall y.(Pxy{\to}Qy)}{\forall y.(Pxy{\to}Qy)}\;\forall\mathbf{E}}{Pxy{\to}Qy}\;\forall\mathbf{E}}{Qy}}{Qy}\;\exists\mathbf{E};1}{(\exists x.Pxy){\to}Qy}\;\exists\mathrm{I};2}{\forall y.((\exists x.Pxy){\to}Qy)}\;\forall\mathrm{I};3$$

Why do we want to mix the two flavours?

In topos logic they *are* the same, and the division is artificial.

We want a system that has 'λ's and in which function terms (like $f(g(x))$) can be constructed step by step.

It may simplify (to humans at least) determining the free variables in each node by inspecting the tree above it.

Subtypes/subobjects: if in a given context we know both x and $\mathrm{prf}[P(x)]$ then in reality that x belongs to a subobject of $\mathbf{E}_x$, the "space of 'x's such that $P(x)$", $\mathbf{E}[x|_{P(x)}] \subset \mathbf{E}_x$; as we are trying to capture formally the intuitive skeletons of proofs it is natural to allow variables ranging over subtypes, as they are just too common in real-world problems.

Topos logic: subobjects are related to the classifier object, and I have some ideas about doing natural deduction in a topos using rules for subobjects. $\omega_{P(x)}$ is the truth-value of $P(x)$; $\top|_{P(x)}$ is a point that asserts that $P(x)$ is true (if, and where, exists; $\mathbf{E}[\top|_{P(x)}]$ may be empty); we have a natural monic $\top|_{P(x)} \hookrightarrow \omega_{P(x)}$; $x|_{P(x)}$ works somewhat like a product $(x, \top|_{P(x)})$, but $\mathbf{E}[\top|_{P(x)}]$ depends on the value of x. I haven't had time time to write the system down and check its details, and the system seems to be easier to formalize in the λ-cube.

A simple deduction in DNC, in full form

$$
\dfrac{
\dfrac{
\dfrac{\mathbf{O}_a \quad \dfrac{\mathbf{C} \quad [\mathbf{O}_x]^3}{\mathbf{O}_x}{:}o \quad [a{\to}x]^1}{a{\to}x}{:}m
}{\mathbf{E}[a{\to}x]}\;\mathbf{E}\mathrm{I};1
\qquad
\dfrac{
\dfrac{
\dfrac{\dfrac{\mathbf{C} \quad [\mathbf{O}_{x'}]^3}{\mathbf{O}_{x'}}{:}o \quad \mathbf{O}_a \quad [a{\to}x']^2}{a{\to}x'}{:}m
\quad
\dfrac{\dfrac{\mathbf{C} \quad [\mathbf{O}_{x'}]^3}{\mathbf{O}_{x'}}{:}o \quad \dfrac{\mathbf{C} \quad [\mathbf{O}_{x''}]^3}{\mathbf{O}_{x''}}{:}o \quad [x'{\to}x'']^3}{x'{\to}x''}{:}m
}{a{\to}x''}
}{(a{\to}x'){\to}(a{\to}x'')}\;2
}{x \Rightarrow' (a{\to}x)}\;{\Rightarrow'}\mathrm{I};3
$$

Its rules:

$$
\dfrac{\mathbf{C} \quad \mathbf{O}_x}{\mathbf{O}_x}{:}o
\qquad
\dfrac{\mathbf{O}_a \quad \mathbf{O}_x \quad a{\to}x}{a{\to}x}{:}m
$$

These rules are like the kstart and tstart rules of λC, but written without the contexts and without the right side of the typing. In '$:o$', if $\mathbf{C}$ is a category, we can introduce a variable $\mathbf{O}_x$, ranging over the objects of $\mathbf{C}$; in '$:m$', if we have two objects of the same category, $\mathbf{O}_a$ and $\mathbf{O}_x$, we can introduce a variable '$a \to x$', ranging over the "set" of morphisms between them.

We are using these rules because we shouldn't be able to abstract a type if there are still free variables with that type around — and a way to control that, in a system where the contexts and typings are left implicit, is to use rules like those and to impose conditions over the discharges: we may only discharge a leaf of a tree if it is the right premiss of one of ':' rules, and if there are no undischarged "terms of that type", as the '$a \to x$' in the tree below:

$$
\dfrac{
\dfrac{\mathbf{O}_a \quad \dfrac{\mathbf{C} \quad [\mathbf{O}_x]^1}{\mathbf{O}_x} \quad a{\to}x}{a{\to}x}{:}m
}{\cdots}\;1 \qquad \leftarrow \text{invalid!}
$$

Composition of morphisms:

$$
\dfrac{a{\to}x' \quad x'{\to}x''}{a{\to}x''}
$$

The usual (functional) abstraction:

$$
\dfrac{\begin{array}{c}\ldots \quad [a{\to}x']^2 \\ \hline\hline a{\to}x''\end{array}}{(a{\to}x'){\to}(a{\to}x'')}\;{\to}\mathrm{I};2
$$

Note that this rule is creating a function from points of an (hom-)"set" to points of other (hom-)"set"; it is only incidental that these hom-sets will be regarded as objects of a category later, and that the function will become a morphism.

More rules for unabbreviated DNC

A rule of introduction for the symbol '**E**':

$$\dfrac{\dfrac{\dots\quad [a{\to}x]^1}{\overline{a{\to}x}}}{\mathbf{E}[a{\to}x]}\;\mathbf{E}I;1 \qquad \dfrac{\dfrac{\dots\quad [\alpha]^1}{\overline{\beta}}}{\mathbf{E}[\beta]}\;\mathbf{E}I;1 \qquad \dfrac{\dfrac{\dots\quad [\mathbf{O}_a]^1\quad [\mathbf{O}_x]^1\quad [a{\to}x]^1}{\overline{a{\to}x}}}{\mathbf{E}[a{\to}x]}\;\mathbf{E}I;1$$

In the first tree an $\mathbf{E}[a \to x]$ — a new type, and also an object of **Set** — is formed by letting the discharged '$a \to x$' range over all its possible values, and packing the results into a set. The second tree is a tentative general form for that rule; if β were $\beta_{f(\alpha)}$ then the $\mathbf{E}[\beta]$ would correspond to $\{\, f(\alpha) \mid \alpha \in \mathbf{E}_\alpha \,\}$.

Note that saying just "$\mathbf{E}[a \to x]$" can be ambiguous. In the first tree above "$\mathbf{E}[a \to x]$" stands for a hom-set, and in the third tree, for all the morphisms in a certain category. I am hoping to come up at some point with a rule that will tell which variables are being abstracted by looking just at the syntax of the newly-formed '**E**'-term; while this doesn't happen we can try to be explicit by saying:

$\mathbf{E}[a \to x] := \mathbf{E}_{(a\to x)}[a \to x]$

or:

$\mathbf{E}[a \to x] := \mathbf{E}_{(\mathbf{O}_a;\mathbf{O}_x;a\to x)}[a \to x]$.

A rule for introduction of the proto-functor symbol:

$$\dfrac{\dfrac{\dots\quad [\mathbf{O}_x]^1}{\overline{\mathbf{E}[a{\to}x]}} \qquad \dfrac{\dots\quad [\mathbf{O}_{x'}]^1\quad [\mathbf{O}_{x''}]^1\quad [x'{\to}x'']^1}{\overline{(a{\to}x'){\to}(a{\to}x'')}}}{x \Rightarrow' (a{\to}x)}\;{\Rightarrow}'\mathrm{I};1$$

or, in the general case:

$$\dfrac{\dfrac{\dots\quad [\mathbf{O}_x]^1}{\overline{\mathbf{O}[x^F]}} \qquad \dfrac{\dots\quad [\mathbf{O}_{x'}]^1\quad [\mathbf{O}_{x''}]^1\quad [x'{\to}x'']^1}{\overline{x'^F{\to}x''^F}}}{x \Rightarrow' x^F}\;{\Rightarrow}'\mathrm{I};1$$

Remember that a proto-functor F from **C** to **D** is (roughly) a pair of operations, (F_o, F_m), that behave "syntactically like a functor": F_o takes objects of **C** to objects of **D**, and F_m takes morphisms of **C** to morphisms of **D** in a way that is compatible with F_o: each $x' \to x''$ goes to a morphism that can be called $x'^F \to x''^F$, i.e., that has right source and the right target; to get a real functor from a proto-functor we need a step like

$$\dfrac{x \Rightarrow' x^F \quad \mathrm{prf}[\Rightarrow]}{x \Rightarrow x^F}$$

where prf[$\Rightarrow$] is a proof of the functoriality of the $x \Rightarrow' x^F$, i.e., a proof that $x \Rightarrow' x^F$ respects identities and compositions.

Representing adjunctions in DNC

An adjunction $L \dashv R$ between two functors $L : \mathbf{A} \to \mathbf{B}$ and $R : \mathbf{B} \to \mathbf{A}$ is written as

$(a^{\mathrm{op}}; b) \overset{\bullet}{\to} ((a^L \to b) \leftrightarrow (a \to b^R))$

or, in a more expressive way (but with the '$(a^{\mathrm{op}}; b) \overset{\bullet}{\to}$' ommited) as:

$$\begin{array}{ccc} a^L & \Leftarrow & a \\ \downarrow & \leftrightarrow & \downarrow \\ b & \Rightarrow & b^R \end{array}$$

Each of the nine entries of that diagram has a precise meaning:

'a^L': $\mathbf{O}[a^L]$ (an object of $\mathbf{B}$) is the image by L of $\mathbf{O}[a]$.
'$\Leftarrow$': $a \Rightarrow a^L$ is the functor L.
'a': $\mathbf{O}[a]$ is an arbitrary object of $\mathbf{A}$.
'$\downarrow$' (left): $a^L \to b$ is a morphism of $\mathbf{B}$.
'$\leftrightarrow$': $(a^L \to b) \leftrightarrow (a \to b^R)$ is a bijection of hom-sets.
'$\downarrow$' (right): $a \to b^R$ is a morphism of $\mathbf{B}$.
'b': $\mathbf{O}[b]$ is an arbitrary object of $\mathbf{B}$.
'$\Rightarrow$': $b \Rightarrow b^R$ is the funtor R.

Note that due to the '$\leftrightarrow$' in the middle if we know $a^L \to b$ (for example, if it is stands for a variable ranging over all the morphisms in a certain hom-set) then we have a default meaning for the $a \to b^R$: make it the image of $a^L \to b$ by the bijection... and the same for the other direction. As we are not writing the term annotations we are not privileging any of the cases — $a^L \to b$ known and $a \to b^R$ defined from it, or the reverse.

The '$\stackrel{\bullet}{\to}$' in the term for an adjunction in DNC

The '$(a^{\rm op};b) \stackrel{\bullet}{\to}$' in '$(a^{\rm op};b) \stackrel{\bullet}{\to} ((a^L \to b) \leftrightarrow (a \to b^R))$' makes the bijection obey the conditions imposed on adjunctions (see for example [BW], p.55). For any morphism $(a^{\rm op};b)' \to (a^{\rm op};b)''$ in $\mathbf{A}^{\rm op} \times \mathbf{B}$ a certain square in **Set** (where the hom-sets live) must commute; restricting our attention to morphisms that are identities in one of the coordinates,

$(a'^{\rm op};b) \to (a''^{\rm op};b)$: it subsumes an $a'^{\rm op} \to a''^{\rm op}$, i.e., an $a' \leftarrow a''$:

$$\begin{array}{ccccc}
(a'^{\rm op};b) & & (a'^L \to b) & \leftrightarrow & (a' \to b^R) \\
\downarrow & & \downarrow & & \downarrow \\
(a''^{\rm op};b) & & (a''^L \to b) & \leftrightarrow & (a'' \to b^R)
\end{array}$$

Intuitively: given either an $a'^L \to b$ or an $a' \to b^R$, both $a''^L \to b$ and $a'' \to b^R$ are well-defined.

$(a^{\rm op};b') \to (a^{\rm op};b'')$: subsumes a $b' \to b''$, and:

$$\begin{array}{ccccc}
(a^{\rm op};b') & & (a^L \to b') & \leftrightarrow & (a \to b'^R) \\
\downarrow & & \downarrow & & \downarrow \\
(a^{\rm op};b'') & & (a^L \to b'') & \leftrightarrow & (a \to b''^R)
\end{array}$$

And intuitively this means that given either an $a^L \to b'$ or an $a \to b'^R$, both $a^L \to b''$ and $a \to b''^R$ are well-defined.

Consider the diagrams below:

$$\begin{array}{ccc}
a''^L & \Leftarrow & a'' \\
\downarrow & \leftarrow & \downarrow \\
a'^L & \Leftarrow & a' \\
\downarrow & \leftrightarrow & \downarrow \\
b' & \Rightarrow & b'^R \\
\downarrow & \to & \downarrow \\
b'' & \Rightarrow & b''^R
\end{array}
\qquad
\begin{array}{ccc}
(a'^L \to b') & \to & (a''^L \to b') \\
\downarrow & \searrow & \downarrow \\
(a'^L \to b'') & \to & (a''^L \to b'')
\end{array}
\qquad
\begin{array}{ccc}
(a' \to b'^R) & \to & (a'' \to b'^R) \\
\downarrow & \searrow & \downarrow \\
(a' \to b''^R) & \to & (a'' \to b''^R)
\end{array}$$

where $a'' \to a'$ and $b' \to b''$ are known, and between each pair of corresponding corners in the two squares at the right there is a bijection, that is not drawn...

These diagrams illustrate ways to start with an $a'^L \to b'$ (or an $a' \to b'^R$) and build an $a'' \to b''^R$ (or an $a'^L \to b'$) from it; all these ways are equivalent.

There are several ways to add term annotations to the operation $(a'^L \to b') \to (a'' \to b''^R)$, but they're all equivalent[2]... so it may be clearer — and, in this case, unambiguous — to refer to this operation by its type only, without choosing any of the equivalent ways to add term annotations to it.

[2] We are just stating this, and in a very informal way; stating this precisely is another story, and proving it is still another.

Minimal contexts for terms of DNC

Another charming thing about DNC is that we can often omit the context for some categorical fact that we are expressing as a term of DNC, and there is a way to reconstruct the context from the term; for example,

$$(a; x \Rightarrow x^F) \overset{\bullet}{\to} ((x \overset{\bullet}{\to} ((a \to x) \to x^F)) \leftrightarrow a^F)$$

"is" the Yoneda lemma; just as there are algorithms for type inference for terms of system HM (i.e., Haskell or ML programs) there are algorithms for type inference and context inference for terms of DNC; they are waiting to be discovered...

Let's do a first at attempt at such an algorithm. It will be conveninet to introduce a new typing symbol, '::', meaning "is an"; we have $a{:}\mathbf{E}_a$ but $a{::}\mathbf{e}[\mathbf{E}_a]$.

If $\mathbf{O}_a$ and $\mathbf{O}_b$ are objects of the same category, then we may speak of '$a \to b$'s; they are morphisms from $\mathbf{O}_a$ to $\mathbf{O}_b$ in that category. That is:

$$\frac{\mathbf{O}_a{::}\mathbf{o}[\mathbf{Cat}_a] \quad \mathbf{O}_b{::}\mathbf{o}[\mathbf{Cat}_b] \quad \mathbf{Cat}_a = \mathbf{Cat}_b}{a{\to}b{::}\mathbf{m}[\mathbf{Cat}_a]}$$

But if we are starting from the big term that is below the bar (the $a \to b$ in this case), about whom we don't know much, and we are tring to understand its type and to construct a context for it, then we may not have a decent ':: ...' part for the $a \to b$ yet, and it is more convenient to use a rule like:

$$\frac{\mathbf{O}_a{::}\mathbf{o}[\mathbf{Cat}_a] \quad \mathbf{O}_b{::}\mathbf{o}[\mathbf{Cat}_b] \quad \mathbf{Cat}_a = \mathbf{Cat}_b}{a{\to}b{::}\mathbf{m}[\mathbf{Cat}_a]}$$

We start with a morphism called $a \to b$; we want to add a ':: ...' part to it, saying that it is a morphism of a certain category; by the definition of the symbol **Cat**, $a \to b$ it is a morphism of $\mathbf{Cat}_a$.

Now if $a \to b$ is a morphism, then $\mathbf{O}_a$ and $\mathbf{O}_b$ are objects; we have default names for the categories where they live, and we also learn that in the given context these categories, $\mathbf{Cat}_a$ and $\mathbf{Cat}_b$, must be the same.

There are some pieces of information that can appear in the "typing and contextualizing trees" for DNC terms that I don't know how to use yet. For example, if we stumble on a functor named $a \Rightarrow a^F$ then we know that '$\mathbf{O}_a$' is a *generic* object of its domain category $\mathbf{Cat}_a$, that is, $\mathbf{O}_a$ may range over all objects of $\mathbf{Cat}_a$. Compare with the case when $\mathbf{O}_x$ and $\mathbf{O}_y$ are objects of **Set**; $\mathbf{O}_x$ and $\mathbf{O}_y$ are generic, but $\mathbf{O}_{(x,y)}$ $(= \mathbf{O}_x \times \mathbf{O}_y)$ is not, because not every object of **Set** is of the form $\mathbf{O}_{(x,y)}$, unless we use a trick. So, $\mathbf{Cat}_{(x,y)} = \mathbf{Set}$, but $\mathbf{O}_{(x,y)}$ is not generic...

Minimal contexts for terms of DNC: Yoneda

Here is a typing/contextualizing tree for the Yoneda lemma; for typographical reasons it was split in two trees with the same shape — the first for what appears at the left of the '::'s, the second for what appears at the right — and the nodes with equalities, like $\mathbf{Cat}_a = \mathbf{Cat}_b$, are not shown.

$$\dfrac{\dfrac{\mathbf{O}_a \quad \dfrac{\dfrac{\mathbf{O}_x \quad \mathbf{O}_{x^F}}{x \Rightarrow x^F}}{\mathbf{O}[x \Rightarrow x^F]}}{\mathbf{O}[(a; x \Rightarrow x^F)]} \quad \dfrac{\dfrac{\dfrac{\dfrac{\mathbf{O}_x \quad \dfrac{\dfrac{\dfrac{\dfrac{\mathbf{O}_a \quad \mathbf{O}_x}{a{\to}x}}{\mathbf{E}[a{\to}x]}}{\mathbf{O}[a{\to}x]} \quad \mathbf{O}_{x^F}}{(a{\to}x){\to}x^F}}{x \overset{\bullet}{\to} ((a{\to}x){\to}x^F)}}{\mathbf{E}[x \overset{\bullet}{\to} ((a{\to}x){\to}x^F)]}}{\mathbf{O}[x \overset{\bullet}{\to} ((a{\to}x){\to}x^F)]} \quad \mathbf{O}_{a^F}}{(x \overset{\bullet}{\to} ((a{\to}x){\to}x^F)){\leftrightarrow}a^F}}{(a; x \Rightarrow x^F) \overset{\bullet}{\to} ((x \overset{\bullet}{\to} ((a{\to}x){\to}x^F)){\leftrightarrow}a^F)}$$

$$\dfrac{\dfrac{\mathbf{O}_a \quad \dfrac{\dfrac{\mathbf{o}[\mathbf{Cat}_x] \quad \mathbf{o}[\mathbf{Cat}_{x^F}]}{\mathbf{o}[\mathbf{Cat}_x \Rightarrow \mathbf{Cat}_{x^F}]}}{\mathbf{o}[\mathbf{Cat}_x \Rightarrow \mathbf{Cat}_{x^F}]}}{\mathbf{o}[\mathbf{Cat}_a \times (\mathbf{Cat}_x \Rightarrow \mathbf{Cat}_{x^F})]} \quad \dfrac{\dfrac{\dfrac{\dfrac{\mathbf{o}[\mathbf{Cat}_x] \quad \dfrac{\dfrac{\dfrac{\dfrac{\mathbf{o}[\mathbf{Cat}_a] \quad \mathbf{o}[\mathbf{Cat}_x]}{\mathbf{m}[\mathbf{Cat}_a]}}{\mathbf{o}[\mathbf{Set}]}}{\mathbf{o}[\mathbf{Set}]} \quad \mathbf{o}[\mathbf{Cat}_{x^F}]}{\mathbf{m}[\mathbf{Set}]}}{\mathbf{m}[\mathbf{Cat}_x \Rightarrow \mathbf{Set}]}}{\mathbf{o}[\mathbf{Set}]}}{\mathbf{o}[\mathbf{Set}]} \quad \mathbf{o}[\mathbf{Cat}_{a^F}]}{\mathbf{m}[\mathbf{Set}]}}{\mathbf{m}[(\mathbf{Cat}_a \times (\mathbf{Cat}_x \Rightarrow \mathbf{Cat}_{x^F})) \Rightarrow \mathbf{Set}]}$$

Note that it has the fragment:

$$\dfrac{\dfrac{\mathbf{E}[x \overset{\bullet}{\to} ((a{\to}x){\to}x^F)]}{\mathbf{O}[x \overset{\bullet}{\to} ((a{\to}x){\to}x^F)]} \quad \mathbf{O}_{a^F}}{(x \overset{\bullet}{\to} ((a{\to}x){\to}x^F)){\leftrightarrow}a^F}$$

The most natural way to make sense of $\mathbf{O}[x \overset{\bullet}{\to} ((a \to x) \to x^F)]$ is to take it as being an object of $\mathbf{Set}$ (and thus an $\mathbf{E}[\ldots]$), the set of all natural transformations between the functors $x \Rightarrow (a \to x)$ and $x \Rightarrow x^F$; as $\mathbf{E}[x \overset{\bullet}{\to} ((a \to x) \to x^F)]$ and $\mathbf{O}[a^F]$ live in the same category, then $\mathbf{Cat}[a^F] = \mathbf{Set}$. A similar thing happens when we make sense of $\mathbf{O}[a \to x]$ by seeing it as an hom-set.

More meaning hidden in DNC terms

> "And only *one* for birthday presents, you know. There's glory for you!"
>
> "I don't know what you mean by 'glory,'" Alice said.
>
> (...)
>
> "That's a great deal to make one word mean," Alice said in a thoughtful tone.
>
> "When I make a word do a lot of work like that," said Humpty Dumpty, "I always pay it extra."

(From *Through the Looking Glass and What Alice Found There*, by Lewis Carroll)

When I write down a term of DNC I mean a lot.

Take the Yoneda lemma:

$$(a; x \Rightarrow x^F) \overset{\bullet}{\to} ((x \overset{\bullet}{\to} ((a \to x) \to x^F)) \leftrightarrow a^F)$$

It is a natural transformation (actually a natural isomorphism) between two functors,
$(a; x \Rightarrow x^F) \Rightarrow (x \overset{\bullet}{\to} ((a \to x) \to x^F))$ and
$(a; x \Rightarrow x^F) \Rightarrow a^F$.
Each of them has a natural construction, and so instead of letting them be arbitrary we set them to the values given by the natural constructions.

Tentative rule: when we don't specify the context it is implicit that the context is taken to be as small as possible — not only it doesn't mention any unneeded symbol, but also all its symbols that can be defined from the others by natural constructions are set to those definitions, instead of being taken as variables. A context may also carry some equalities — in the Yoneda lemma, for example, $\mathbf{Cat}_a = \mathbf{Cat}_x$ and $\mathbf{Cat}[a^F] = \mathbf{Set}$. We don't want to have more of these equalities than the minimum necessary; a "good" context for the Yoneda lemma can't force $\mathbf{Cat}_a = \mathbf{Set}$. How to formalize all that? What are the universality conditions that a good context satisfies?

If we say "*the* Yoneda lemma", in a situation where our definition of the Yoneda lemma is its DNC term, then it is implicit that the "meaning" of that term, as a natural isomorphism (depending only on the category $\mathbf{Cat}_a = \mathbf{Cat}_x$) is well-defined; it will also mean the each of the functors on top of which the Yoneda lemma is built is also well-defined, in the same sense. That is, the Yoneda lemma and all of its "subterms" lie in a kind of a "coherent fragment" of DNC... but we need to formalize (and prove!) that.

When we say "an adjunction is a natural isomorphism $(a^{\mathrm{op}}; b) \overset{\bullet}{\to} ((a^L \to b) \leftrightarrow (a \to b^R))$" it is implicit that to define an adjunction formally we need to specify $\mathbf{Cat}_a$, $\mathbf{Cat}_b$, $a \Rightarrow a^L$ and $b \Rightarrow b^R$...

For me it is clear that terms of DNC do have glory, where "glory" is defined as in the Humpty Dumpty chapter of "Throught the Looking Glass", as meaning "a nice knock-down argument". But I have to pay the words extra.

To be continued...