Table of Contents

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`

Note: I'm currently at McGill university, as a visitor — I'm doing a "sandwich PhD" (really!) thanks to a CAPES (`http://www.capes.gov.br/`) grant...

Abstract

(For the FMCS talk)

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

Even though derivations in DNC tell only half of the story, they still have a certain charm: DNC terms represent "types", but a tree represents a construction of a lambda-calculus term; there's a Curry-Howard isomorphism going on, and a tree can be a visual help for understanding how the lambda-calculus term works — how the data flows inside a given categorical construction. Also, if we are looking for a categorical entity of a certain "type" we can try to express it as a DNC term, and then look for a DNC "deduction" having it as its "conclusion"; the deduction will give us a T-part, and we will still have to go back to the standard language to supply a P-part, but at least the search has been broken in two parts...

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.

The "generic point" notation

What would be the natural name for a variable in...

A	$\dashrightarrow$	a	$A = \mathbf{E}[a]$
B	$\dashrightarrow$	b	$B = \mathbf{E}[b]$
$A \times B$	$\dashrightarrow$	a, b	$A \times B = \mathbf{E}[a, b] = \mathbf{E}[a] \times \mathbf{E}[b]$
B^A	$\dashrightarrow$	$a \mapsto b$	$B^A = \mathbf{E}[a \mapsto b] = \mathbf{E}[b]^{\mathbf{E}[a]}$

We will allow strange names for variables and terms
and we will have a dictionary to translate these names
into something precise (in the Calculus of Constructions).

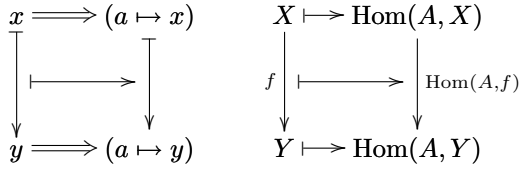
'**E**' is pronounced as "the space of".

Functors and NTs in this notation

Choose sets A and B, and a function $B \xrightarrow{g} A$.

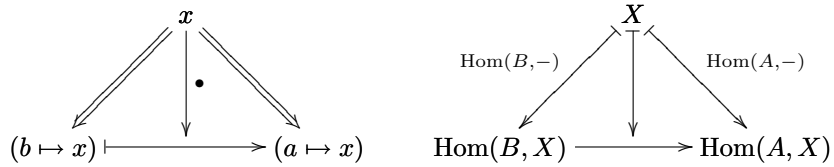
In GP/DNC notation we write $\mathrm{Hom}(A, -)$ as $x \Rightarrow (a \mapsto x)$.

Note that from the name "$x \Rightarrow (a \mapsto x)$" we can extract both the action on objects, $\mathbf{E}[x] \mapsto \mathbf{E}[a \mapsto x]$, and the action on morphisms, $(x \mapsto y) \mapsto ((a \mapsto x) \mapsto (a \mapsto y))$.



$x \xrightarrow{\bullet} ((b \mapsto x) \mapsto (a \mapsto x))$ is a natural transformation going from $x \Rightarrow (b \mapsto x)$ to $x \Rightarrow (a \mapsto x)$.

It works as $\mathbf{E}[x] \mapsto ((b \mapsto x) \mapsto (a \mapsto x))$, but it also obeys a naturality condition...



What is a functor?
A functor $F : \mathbf{A} \to \mathbf{B}$ is a package containing:

$$\begin{array}{ll} F_O & F_O A : \mathrm{Objs}_{\mathbf{B}} \\ F_M & F_M A' A''(A' \xrightarrow{f} A'') : \mathrm{Hom}_{\mathbf{B}}(F_O(A'), F_O(A'')) \\ & F_M A' A''(A' \xrightarrow{f} A'') = (F_O(A') \xrightarrow{F(f)} F_O(A'')) \end{array}$$

and two "proofs":
• that F respects composition, $F_M(k \circ h) = F_M(k) \circ F_M(h)$
• that F respects identities, $F_M(\mathrm{id}_A) = \mathrm{id}_{F_O(A)}$.

A proto-functor only has F_O and F_M.
The prefix "proto" will always mean "*without the terms for equalities*".

A category $\mathbf{A}$ is a package containing:

$$\begin{array}{ll} \mathrm{Objs}_{\mathbf{A}} & \\ \mathrm{Hom}_{\mathbf{A}} & \mathrm{Hom}_{\mathbf{A}}\, A' A'' \\ \mathrm{id}_{\mathbf{A}} & \mathrm{id}_{\mathbf{A}} A : \mathrm{Hom}_{\mathbf{A}}\, AA \\ \circ_{\mathbf{A}} & \circ_{\mathbf{A}} A' A'' A'''(A' \xrightarrow{h} A'')(A'' \xrightarrow{k} A''') : \mathrm{Hom}_{\mathbf{A}}\, A' A''' \end{array}$$

plus three "proofs" that say that $\circ_{\mathbf{A}}$ is associative and respects $\mathrm{id}_{\mathbf{A}}$.
A proto-category only has $\mathrm{Objs}_{\mathbf{A}}$, $\mathrm{Hom}_{\mathbf{A}}$, $\mathrm{id}_{\mathbf{A}}$, $\circ_{\mathbf{A}}$,

Toy example: $G \circ F$

$$\dfrac{\dfrac{\dfrac{[\mathbf{O}[a]]^1 \quad a \Rightarrow a^F}{\mathbf{O}[a^F]} \quad b \Rightarrow b^G}{\mathbf{O}[a^{FG}]}\ 1}{\mathbf{O}[a] \mapsto \mathbf{O}[a^{FG}]} \qquad \dfrac{\dfrac{\dfrac{[A]^1 \quad F}{F(A)} \quad G}{G(F(A))}\ 1}{\lambda A.G(F(A))}$$

$$\dfrac{\dfrac{\dfrac{[\mathbf{O}[a]]^1 \quad a \Rightarrow a^F}{(a \Rightarrow a^F)(\mathbf{O}[a^F])} \quad b \Rightarrow b^G}{(b \Rightarrow b^G)((a \Rightarrow a^F)(\mathbf{O}[a^F]))}\ 1}{\lambda \mathbf{O}[a].(b \Rightarrow b^G)((a \Rightarrow a^F)(\mathbf{O}[a^F]))}$$

$$(a \Rightarrow a^F G)_O := \lambda \mathbf{O}[a]{:}\mathrm{Objs}_{\mathbf{A}}.(b \Rightarrow b^G)_O((a \Rightarrow a^F)_O(\mathbf{O}[a^F]))$$

$$\dfrac{\dfrac{\dfrac{[a' \mapsto a'']^1 \quad a \Rightarrow a^F}{a'^F \mapsto a''^F} \quad b \Rightarrow b^G}{a'^{FG} \mapsto a''^{FG}}\ (\Rightarrow E); 1}{a \Rightarrow a^{FG}} \qquad \dfrac{\dfrac{h \quad F}{F(h)} \quad G}{\dfrac{G(F(h))}{\lambda h.G(F(h))}}$$

$$\dfrac{\dfrac{\dfrac{[a' \mapsto a'']^1 \quad a \Rightarrow a^F}{(a \Rightarrow a^F)(a' \mapsto a'')} \quad b \Rightarrow b^G}{(b \Rightarrow b^G)((a \Rightarrow a^F)(a' \mapsto a''))}}{\lambda(a' \mapsto a'').(b \Rightarrow b^G)((a \Rightarrow a^F)(a' \mapsto a''))}\ 1$$

$$\begin{aligned} F_M := {} & \lambda \mathbf{O}[a']{:}\mathrm{Objs}_{\mathbf{A}}.\lambda \mathbf{O}[a'']{:}\mathrm{Objs}_{\mathbf{A}}. \\ & \lambda(a' \mapsto a''){:}(\mathrm{Hom}_{\mathbf{A}}\, \mathbf{O}[a']\mathbf{O}[a'']). \\ & (b \Rightarrow b^G)_M((a \Rightarrow a^F)_O \mathbf{O}[a'])((a \Rightarrow a^F)_O \mathbf{O}[a'']) \\ & ((a \Rightarrow a^F)_M \mathbf{O}[a']\mathbf{O}[a''](a' \mapsto a'')) \end{aligned}$$

$$\dfrac{\dfrac{\dfrac{a' \mapsto a'' \quad a \Rightarrow a^F}{a'^F \mapsto a''^F} \quad b \Rightarrow b^G}{a'^{FG} \mapsto a''^{FG}} \qquad \dfrac{\dfrac{a'' \mapsto a''' \quad a \Rightarrow a^F}{a''^F \mapsto a'''^F} \quad b \Rightarrow b^G}{a''^{FG} \mapsto a'''^{FG}}}{a'^{FG} \mapsto a'''^{FG}} \qquad \dfrac{\dfrac{h \quad F}{F(h)} \quad G}{G(F(h))} \quad \dfrac{\dfrac{k \quad F}{F(k)} \quad G}{G(F(k))}}{G(F(k)) \circ G(F(h))}$$

$$\dfrac{\dfrac{\dfrac{a' \mapsto a'' \quad a'' \mapsto a'''}{a' \mapsto a'''} \quad a \Rightarrow a^F}{a'^F \mapsto a'''^F} \quad b \Rightarrow b^G}{a'^{FG} \mapsto a'''^{FG}} \qquad \dfrac{\dfrac{h \quad k}{k \circ h} \quad F}{\dfrac{F(k \circ h)}{G(F(k \circ h))} \quad G}$$

Why DNC? Two conjectures

(name → derivation → λ-term...)

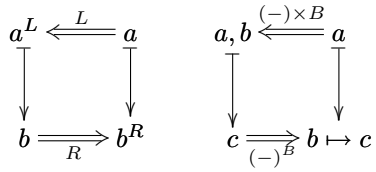
Conjecture 1: the naturality conditions do come for free.
Take a name for a functor — say, $a \Rightarrow a^{FG}$, with hyps. $a \Rightarrow a^F$ and $b \Rightarrow b^G$ — and a natural construction for a (proto) entity with that name.
Then that construction can be lifted from the syntactical world to the real world — i.e., we can "prove" the missing "prf" terms.

Conjecture 2: given a name, we can effective obtain the natural entity with that name, or prove that it doesn't exist, or that it exists but is not unique.
That is, we have an algorithm for proof-search that does that; and maybe also some purely syntactical criteria that can say "no natural entity with that name can exist" or "there will be non-equivalent constructions".

Adjunctions as DNC diagrams

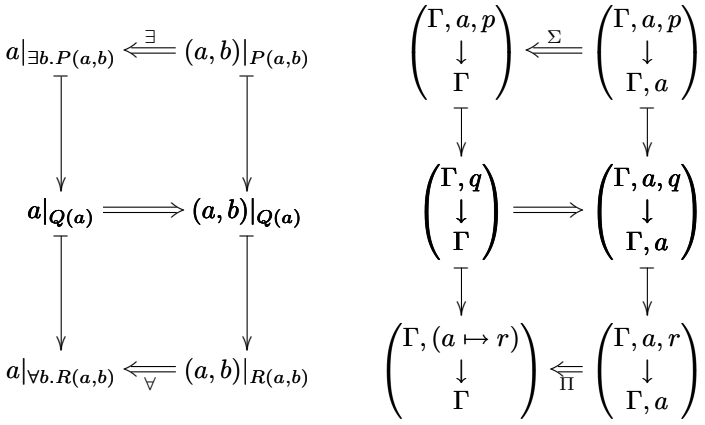
$L \dashv R$ is $(a; b) \xrightarrow{\bullet} ((a^L \mapsto b) \leftrightarrow (a \mapsto b^R))$

$(-) \times B \dashv (-)^B$ is $(a; c) \xrightarrow{\bullet} ((a, b \mapsto c) \leftrightarrow (a \mapsto (b \mapsto c)))$



The functor from $\mathrm{Sub}(A)$ to $\mathrm{Sub}(A \times B)$ (A, B sets, for example) has a left adjoint $\exists$ and a right adjoin $\forall$.

The weakening functor induced by $\Gamma, a \mapsto \Gamma$ in a "good" codomain fibration has a left adjoint Σ and a right adjoint Π.



The proof of the Yoneda Lemma in DNC

$$\dfrac{\dfrac{\mathbf{O}[a] \quad [\mathbf{O}[x]]^1}{\mathbf{E}[a \mapsto x]}\ \mathrm{Hom}_{\mathbf{C}}}{\mathbf{O}[x] \mapsto \mathbf{E}[a \mapsto x]}\ 1 \qquad \dfrac{\dfrac{[a \mapsto x']^1 \quad [x' \mapsto x'']^2}{a \mapsto x''}}{\dfrac{(a \mapsto x') \mapsto (a \mapsto x'')}{x \Rightarrow (a \mapsto x)}\ 1}\ (\Rightarrow I); 2$$

$$\dfrac{\dfrac{\mathbf{O}[a] \quad \dfrac{\dfrac{\dfrac{\mathbf{O}[a]}{x \Rightarrow (a \mapsto x)}}{\mathbf{O}[x \Rightarrow (a \mapsto x)]}\ \mathrm{ren} \quad \mathbf{O}[x \Rightarrow x^F]}{\mathbf{E}[x \xrightarrow{\bullet} ((a \mapsto x) \mapsto x^F)]}\ \mathrm{Hom}_{\mathbf{Set}^{\mathbf{C}}}}{x \xrightarrow{\bullet} ((a \mapsto x) \mapsto x^F)}\ [s]^1}{(a \mapsto a) \mapsto a^F} \quad \dfrac{\mathbf{O}[a]}{a \mapsto a}}{a^F}}{(x \xrightarrow{\bullet} ((a \mapsto x) \mapsto x^F)) \mapsto a^F}\ 1$$

$$\dfrac{\dfrac{\dfrac{\mathbf{O}_a \quad \dfrac{\mathbf{O}[x \Rightarrow x^F]}{x \Rightarrow x^F}\ \mathrm{ren}}{\dfrac{\mathbf{E}[a^F]}{a^F}\ [s]^3} \quad \dfrac{\dfrac{\mathbf{O}_a \quad [\mathbf{O}_x]^2}{\mathbf{E}[a \mapsto x]}}{a \mapsto x}\ [s]^1 \quad \dfrac{\mathbf{O}[x \Rightarrow x^F]}{x \Rightarrow x^F}\ \mathrm{ren}}{a^F \mapsto x^F}}{x^F}}{(a \mapsto x) \mapsto x^F}\ 1}{x \xrightarrow{\bullet} ((a \mapsto x) \mapsto x^F)}\ 2}{a^F \mapsto (x \xrightarrow{\bullet} ((a \mapsto x) \mapsto x^F))}\ 3$$

$$\dfrac{\dfrac{\dfrac{[\mathbf{O}_a]^1 \quad [\mathbf{O}[x \Rightarrow x^F]]^1}{a^F \mapsto (x \xrightarrow{\bullet} ((a \mapsto x) \mapsto x^F))} \quad \dfrac{[\mathbf{O}_a]^1 \quad [\mathbf{O}[x \Rightarrow x^F]]^1}{(x \xrightarrow{\bullet} ((a \mapsto x) \mapsto x^F)) \mapsto a^F}}{a^F \leftrightarrow (x \xrightarrow{\bullet} ((a \mapsto x) \mapsto x^F))}}{(a; x \Rightarrow x^F) \xrightarrow{\bullet} (a^F \leftrightarrow (x \xrightarrow{\bullet} ((a \mapsto x) \mapsto x^F)))}\ 1$$

The proof of the Yoneda Lemma in DNC (2)

...and if we want to prove that the functors implicit in the outer '$\xrightarrow{\bullet}$' of $(a; x \Rightarrow x^F) \xrightarrow{\bullet} (a^F \leftrightarrow (x \xrightarrow{\bullet} ((a \to x) \to x^F)))$ are really (proto!) functors, we need this:

$$\dfrac{\dfrac{[a' \mapsto a'']^1 \qquad \dfrac{[(x \Rightarrow x^{F'}) \mapsto (x \Rightarrow x^{F''})]^1}{x \xrightarrow{\bullet} (x^{F'} \mapsto x^{F''})}\ \text{ren}}{a'^{F'} \mapsto a''^{F''}}}{(a; x \Rightarrow x^F) \Rightarrow a^F}\ 1$$

$$\dfrac{\dfrac{\dfrac{\dfrac{[\mathbf{O}_a]^1}{x \Rightarrow (a \mapsto x)}}{\mathbf{O}[x \Rightarrow (a \mapsto x)]}\ \text{ren} \qquad [\mathbf{O}[x \Rightarrow x^F]]^1}{\mathbf{E}[x \xrightarrow{\bullet} ((a \mapsto x) \mapsto x^F)]}\ \text{Hom}_{\mathbf{Set}^{\mathbf{C}}}}{\mathbf{O}[a; x \Rightarrow x^F] \mapsto \mathbf{E}[x \xrightarrow{\bullet} ((a \mapsto x) \mapsto x^F)]}\ 1$$

$$\dfrac{\dfrac{\dfrac{\dfrac{[a' \mapsto a'']^4 \quad [a'' \mapsto x]^1}{a' \mapsto x} \quad \dfrac{[\mathbf{O}_x]^2 \quad [x \xrightarrow{\bullet} ((a' \mapsto x) \mapsto x^{F'})]^3}{(a' \mapsto x) \mapsto x^{F'}}}{x^{F'}} \quad \dfrac{[\mathbf{O}_x]^2 \quad [x \xrightarrow{\bullet} (x^{F'} \mapsto x^{F''})]^4}{x^{F'} \mapsto x^{F''}}}{x^{F''}}}{(a'' \mapsto x) \mapsto x^{F''}}\ 1}{x \xrightarrow{\bullet} ((a'' \mapsto x) \mapsto x^{F''})}\ (\xrightarrow{\bullet} I); 2}{(x \xrightarrow{\bullet} ((a' \mapsto x) \mapsto x^{F'})) \mapsto (x \xrightarrow{\bullet} ((a'' \mapsto x) \mapsto x^{F''}))}\ 3}{(a; x \Rightarrow x^F) \Rightarrow (x \xrightarrow{\bullet} ((a \mapsto x) \mapsto x^F))}\ (\Rightarrow I); 4$$

A crash course on the Calculus of Constructions
A countable set of *variables*, e.g. $\{a, b, c, \dots, A, B, C, \dots\}$.
Sorts: $*$, $\Box$.
Axioms on sorts: $\vdash *:\Box$.
From that we define *pre-terms*, *pre-judgements* and (valid) derivations.
Terms and judgements are pre-terms and pre-judgements that
can appear in valid derivations.

Example of a term, with a typing: $(\lambda X{:}* .x{:}X.x){:}(\Pi X{:}* .\Pi x{:}X.X)$
(the polymorphic identity)

"$\lambda a{:}A.b$":
λ	a function that expects a value
a	for the variable a
$:A$	(it must be a member of A)
$.b$	and returns b

"$\Pi a{:}A.B$":
Π	the space of the functions that expect
$a{:}A$	"a"s (members of A)
B	and return for each one a member of B

We can have dependent types! See the example above.

A crash course on CC: (pre-)judgments
"$x_1{:}A_1, x_2{:}A_2, \ldots, x_n{:}A_n \vdash t{:}T$":
If we have chosen a value for x_1 (a member of A_1),
a value for x_2 (a member of A_2), etc,
then we know the value of t (a member of T).

Fact:
"$x_1{:}A_1(:S_1), \ldots, x_n{:}A_n(:S_n) \vdash t{:}T$":
Every variable is two levels below a sort.
Every term is either a sort, or one or two levels below a sort.

$$\begin{array}{ccccccc} & & \text{operator} & : & \text{kind} & : & \Box \\ \text{element} & : & \text{set} & : & * & & \end{array}$$

If we know the "class" of each variable we know the class
of every term. Trick:
$(\lambda a^{R\text{-}2}{:}A^{R\text{-}1}.b^{S\text{-}2})^{S\text{-}2}$ $\quad$ $(f^{S\text{-}2}a^{R\text{-}2})^{S\text{-}2}$
$(\Pi a^{R\text{-}2}{:}A^{R\text{-}1}.B^{S\text{-}1})^{S\text{-}1}$

The rules of CC have "variations" for operating on different levels:

$$\begin{array}{ccccccccc|l}
a, a' & : & A & : & * & & & & \Pi_{**}, \\
b, fa' & : & B & : & * & & & & \lambda_{**}, \\
f, (\lambda a{:}A.b) & : & (\Pi a{:}A.B) & : & * & & & & \text{app}_{**} \\
\hline
 & & a, a' & : & A & : & \Box & & \Pi_{\Box *}, \\
b, fa' & : & B & : & * & & & & \lambda_{\Box *}, \\
f, (\lambda a{:}A.b) & : & (\Pi a{:}A.B) & : & * & & & & \text{app}_{\Box *} \\
\hline
a, a' & : & A & : & * & & & & \Pi_{*\Box}, \\
 & & b, fa' & : & B & : & \Box & & \lambda_{*\Box}, \\
 & & f, (\lambda a{:}A.b) & : & (\Pi a{:}A.B) & : & \Box & & \text{app}_{*\Box} \\
\hline
 & & a, a' & : & A & : & \Box & & \Pi_{\Box\Box}, \\
 & & b, fa' & : & B & : & \Box & & \lambda_{\Box\Box}, \\
 & & f, (\lambda a{:}A.b) & : & (\Pi a{:}A.B) & : & \Box & & \text{app}_{\Box\Box}
\end{array}$$

A crash course on CC: the rules

Rules of the system: ax, s, Π, λ, app, weak, conv
(conv is not shown; if $a{:}A$ and $A =_\beta A'$ then $a{:}A'$)

$$\frac{\Gamma \vdash A{:}R \quad \Gamma, a{:}A \vdash B{:}S}{\Gamma \vdash (\Pi a{:}A.B){:}S}\ \Pi_{RS} \qquad \frac{}{\vdash *{:}\Box}\ \text{ax}$$

$$\frac{\Gamma, a{:}A \vdash b{:}B \quad \Gamma \vdash (\Pi a{:}A.B){:}S}{\Gamma \vdash (\lambda a{:}A.b){:}(\Pi a{:}A.B)}\ \lambda_{RS} \qquad \frac{\Gamma \vdash A{:}R}{\Gamma, a{:}A \vdash a{:}A}\ \text{s}_R$$

$$\frac{\Gamma \vdash f{:}(\Pi a{:}A.B) \quad \Gamma \vdash a'{:}A}{\Gamma \vdash fa'{:}B[a := a']}\ \text{app}_{RS} \qquad \frac{\Gamma, \Delta \vdash c{:}C \quad \Gamma \vdash A{:}R}{\Gamma, a{:}A, \Delta \vdash c{:}C}\ \text{weak}_R$$

R, S sorts; a variable; Γ, Δ contexts
All other letters (a', A, b, B, c, C, f) represent terms.
A derivation in CC:

$$\dfrac{\dfrac{\dfrac{\vdash *{:}\Box}{X{:}* \vdash X{:}*}\ \text{s}_\Box \qquad \dfrac{\dfrac{\vdash *{:}\Box \quad \vdash *{:}\Box}{X{:}* \vdash *{:}\Box}\ \text{weak}_\Box \qquad \dfrac{\vdash *{:}\Box}{X{:}* \vdash X{:}*}\ \text{s}_\Box}{X{:}*, x{:}X \vdash *{:}\Box}\ \text{weak}_*}{X{:}* \vdash (\Pi x{:}X.*){:}\Box}\ \Pi_{*\Box}}{X{:}*, F{:}(\Pi x{:}X.*) \vdash F{:}(\Pi x{:}X.*)}\ \text{s}_\Box$$

Most of the tree is "bureaucracy" (weakings, for example)

$$\underbrace{x_1 \overbrace{:A_1}^{\text{typing}}, \ldots, x_n \overbrace{:A_n}^{\text{typing}}}_{\text{context}} \vdash t \overbrace{:T}^{\text{typing}}$$

Fact: (abbreviated CC + dictionary) formalizes DNC
Take a "good" tree with minimal contexts, for example,

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash *:\Box}}{I:* \vdash I:*}\,[\mathrm{s}_\Box]^{C2} \quad \dfrac{}{\vdash *:\Box}}{I:* \vdash (\Pi i{:}I.*):\Box}\,\Pi_{*\Box} \quad \dfrac{\dfrac{}{\vdash *:\Box}}{I:*, i{:}I \vdash i:I}\,[\mathrm{s}_*]^2}{I:*, A{:}(\Pi i{:}I.*) \vdash A:(\Pi i{:}I.*)}\,\mathrm{s}_\Box}{I:*, A{:}(\Pi i{:}I.*), i{:}I \vdash Ai:*}\,\mathrm{app}_{*\Box};1}{I:*, A{:}(\Pi i{:}I.*) \vdash (\Pi i{:}I.Ai):*}\,\Pi_{**};2}{I:*, A{:}(\Pi i{:}I.*), f{:}(\Pi i{:}I.Ai) \vdash f:(\Pi i{:}I.Ai)}\,s_*$$

erase its contexts, and put in the dictionary all the remaining typings:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{*:\Box}}{I:*}\,[\mathrm{s}_\Box]^{C2} \quad \dfrac{}{*:\Box}}{(\Pi i{:}I.*):\Box}\,\Pi_{*\Box} \quad \dfrac{\dfrac{}{*:\Box}}{i:I}\,[\mathrm{s}_*]^2}{A:(\Pi i{:}I.*)}\,\mathrm{s}_\Box}{Ai:*}\,\mathrm{app}_{*\Box};1}{(\Pi i{:}I.Ai):*}\,\Pi_{**};2}{f:(\Pi i{:}I.Ai)}\,s_*$$

Dictionary:
$i : I : *$
$A : (\Pi i{:}I.*) : \Box$
$Ai : *$
$f : (\Pi i{:}I.Ai) : *$

...now erase the typings too:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{*}}{I}\,[\mathrm{s}_\Box]^{C2} \quad \dfrac{}{*}}{\Pi i{:}I.*}\,\Pi_{*\Box} \quad \dfrac{\dfrac{}{*}}{I}\,[\mathrm{s}_*]^2}{A}\,\mathrm{s}_\Box}{Ai}\,\mathrm{app}_{*\Box};1}{\Pi i{:}I.Ai}\,\Pi_{**};2}{f}\,s_*$$

Dictionary:
$i : I : *$
$A : (\Pi i{:}I.*) : \Box$
$Ai : *$
$f : (\Pi i{:}I.Ai) : *$

Fact: it is possible to reconstruct the original tree from this one.
It is also possible to discard the dictionary and reconstruct it just from that last tree.

Fact: (abbreviated CC + dictionary) formalizes DNC (2)

$$\dfrac{\dfrac{\dfrac{}{*}}{I}\,s_\square \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{*}}{I}\;[\mathrm{s}_\square]^{C2} \quad \dfrac{}{*}}{\Pi i{:}I.*}\,\Pi_{*\square}}{A}\,\mathrm{s}_\square \qquad \dfrac{\dfrac{\dfrac{}{*}}{I}\;[\mathrm{s}_\square]^{C1}}{i}\,[\mathrm{s}_*]^2}{Ai}\,\mathrm{app}_{*\square};1}{\dfrac{\Pi i{:}I.Ai}{f}\,s_*}\,\Pi_{**};2$$

Dictionary:
$i : I : *$
$A : (\Pi i{:}I.*) : \square$
$Ai : *$
$f : (\Pi i{:}I.Ai) : *$

Next step: replace the terms in a tree with names for them in the GP notation; let the dictionary translate some terms as applications — example, $b \dashrightarrow (a \mapsto b)(a)$.

$$\dfrac{\dfrac{\dfrac{}{*}}{\mathbf{E}[i]}\,s_\square \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{*}}{\mathbf{E}[i]}\;[\mathrm{s}_\square]^{C2} \quad \dfrac{}{*}}{\mathbf{E}[i \mapsto \mathbf{e}*]}\,\Pi_{*\square}}{i \mapsto \mathbf{E}[a_i]}\,\mathrm{s}_\square \qquad \dfrac{\dfrac{\dfrac{}{*}}{\mathbf{E}[i]}\;[\mathrm{s}_\square]^{C1}}{i}\,[\mathrm{s}_*]^2}{\mathbf{E}[a_i]}\,\mathrm{app}_{*\square};1}{\dfrac{\mathbf{E}[i \mapsto a_i]}{i \mapsto a_i}\,s_*}\,\Pi_{**};2$$

$$\begin{array}{ll} & \text{Dictionary:} \\ & i : \mathbf{E}[i] : * \\ & i \mapsto \mathbf{E}[a_i] : \mathbf{E}[i \mapsto \mathbf{e}*] : \square \\ \mathbf{E}[a_i] = & (i \mapsto \mathbf{E}[a_i])(i) : * \\ & i \mapsto a_i : \mathbf{E}[i \mapsto a_i] : * \end{array}$$

New variables are introduced only at the "s" rules.
To help keep track of what's happening, we mark discharges with '$[\cdot]^n$'s and "contractions" with '$[\cdot]^{Cn}$'s.

We can split the tree at the "s" rules — "hypotheses" become entities obtained by "s" rules.

Protocategories
(Again: the prefix "*proto*" always means "*without the terms for equalities*").

A protocategory $\mathbf{C} : \mathrm{Protocats}_{SR}$
(where S, R are sorts — typically $S = \Box$, $R = *$)
is a package containing

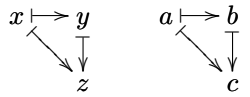
$\mathrm{Objs}_{\mathbf{C}} : S$	
$\mathrm{Hom}_{\mathbf{C}}$	$\mathrm{Hom}_{\mathbf{C}}\, AB : R$
$\mathrm{id}_{\mathbf{C}}$	$\mathrm{id}_{\mathbf{C}} A : \mathrm{Hom}_{\mathbf{C}}\, AA$
$\circ_{\mathbf{C}}$	$\circ_{\mathbf{C}} ABC(a \mapsto b)(b \mapsto c) : \mathrm{Hom}_{\mathbf{C}}\, AC$

$(A, B, C : \mathrm{Objs}_{\mathbf{C}})$

Notational trick: $a \mapsto b : \mathbf{E}[a \mapsto b]$,
and the dictionary translates $\mathbf{E}[a \mapsto b]$ $\dashrightarrow$ $\mathrm{Hom}_{\mathbf{C}}\, AB$.

A very special example: **Set** is a $\Box*$-protocategory.
$\mathrm{Objs}_{\mathbf{Set}} = * : \Box$
$\mathrm{Hom}_{\mathbf{Set}} := \lambda X{:}*\,.\lambda Y{:}*\,.(\Pi x{:}X.Y)$
$\mathrm{id}_{\mathbf{Set}} := \lambda X{:}*\,.\lambda x{:}X.x$
$\circ_{\mathbf{Set}} := \lambda X{:}*\,.\lambda Y{:}*\,.\lambda Z{:}*\,.$
$\lambda f{:}(\mathrm{Hom}_{\mathbf{Set}}\, XY).\lambda g{:}(\mathrm{Hom}_{\mathbf{Set}}\, YZ).$
$\lambda x{:}X.g(fx)$

Protocategories: "O", "Cat", "$\mapsto$" as morphism
A typical $\Box*$-protocategory:
$\mathrm{Objs}_{\mathbf{C}} : \Box$
$\mathrm{Hom}_{\mathbf{C}} : \Pi\mathbf{O}[a]{:}\mathrm{Objs}_{\mathbf{C}}.\Pi\mathbf{O}[b]{:}\mathrm{Objs}_{\mathbf{C}}.*$
$\mathbf{O}[a], \mathbf{O}[b], \mathbf{O}[c] : \mathrm{Objs}_{\mathbf{C}}$
like $\mathbf{E}[x], \mathbf{E}[y], \mathbf{E}[z] : \mathrm{Objs}_{\mathbf{Set}} = *$
but <u>we don't have a semantics</u> for "a", "b", "c"!!!
Just for $\mathbf{O}[a]$, $\mathbf{O}[b]$, $\mathbf{O}[c]$, $(a \mapsto b)$, $(b \mapsto c)$, $(a \mapsto c)$.



in **Set** in **C**

New "operator" in the dictionary: **Cat** —
$\mathbf{Cat}[x]$ --→ **Set**, $\mathbf{Cat}[a]$ --→ **C** ($\mathbf{O}[a]{:}\mathrm{Objs}_{\mathbf{C}}$)

$x \mapsto y : \mathbf{E}[x \mapsto y] = \mathbf{E}[x] \to \mathbf{E}[y]$
$a \mapsto b : \mathbf{E}[a \mapsto b] = \mathrm{Hom}_{\mathbf{C}}\, \mathbf{O}[a]\mathbf{O}[b]$

A crash course on CC: propositions and proofs
prop[P] is P seen as a proposition.
Naively, prop[P] is the set of proofs of P.
prf[P] : prop[P] : $*$

Dictionary:

prop[$P \supset Q$]	--→	(Πprf[P]:prop[P].prop[Q])
prf[$P \supset Q$]	--→	(prf[P] $\mapsto$ prf[Q])
prop[$\forall x.P(x)$]	--→	(Πx:**E**[x].prop[$P(x)$])
prf[$\forall x.P(x)$]	--→	($x \mapsto$ prf[$P(x)$])

To speak "internally" about equality between members of the same set we need a package of terms, that will be added to contexts...

equality$_*$ is a package consisting of:

eq$_*$	(eq$_* Aaa'$) : Props
	prop[$a = a'$] --→ eq$_* Aaa'$
refl$_*$	(refl$_* Aa$) : prop[$a = a$]
	prf[$a = a$] --→ refl$_* Aa$
sim$_*$	(sim$_* Aaa'$) : prop[$a = a' \supset a' = a$]
	prf[$a = a' \supset a' = a$] --→ (sim$_* Aaa'$)
trans$_*$	(trans$_* Aaa'a''$) : prop[$a = a' \supset (a' = a'' \supset a = a'')$]
	prf[$a = a' \supset (a' = a'' \supset a = a'')$] --→ (trans$_* Aaa'a''$)

f′s.r.equality$_{**}$ is a package of terms saying that
all functions f:(Πa:A.B) "respect equality".