2018-01-16

15-819 : Computational Type Theory

Plan: Contrast of CTT vs Formal TT by means of simple type theory. Historical context and higher Type theory. The analysis of higher ~~order~~ type theory will require both CTT and FTT.

(In some order:

0. fundamentals of CTT (in zero dim's)
1. RedPRL implementation of CHiTT
2. guarded ~~CTT~~ compl'l type theory.
3. higher type theory / cubical type theory
   a) cubical infrastructure
   b) univalence, inductive types.
4. CTT as a spec language for ~~pro~~ prog verification.
   ⇒ types are more than just propositions! Down with Curry-Howard!
   TYPE Theory as a theory of proof (CH)
   vs as a theory of truth (BHK)
   Type theory accounts for more than just proofs.
5. Comparison with various higher FTTs:
   CCHM, ABCFHL (cartesian form).
   ~~(Klees~~ (Kleene algebra form)

:

eg) Nanevski (IMDEA)'s type theory for concurrency, which seems to be computational, vs session types, which are formal/C-H motivated.

Formal TT

aka structural
prescriptive
axiomatic
formal logic.

You're defining an R.E. st.
Inductively defining
"judgments" (RWH thinks
this is regrettable naming).

$\Gamma \vdash A$ type } defining terms
$\Gamma \vdash M : A$ } and types.

~~Gross $x_1 : A_1, \ldots, x_n : A_n$~~

Defined by a collection of rules. Often equipped with a notion of equality ("definitional equality")

$\Gamma \vdash A \equiv B$, $\Gamma \vdash M \equiv N : A$

that often captures comp, where typing, etc, are stable under definitional equality:

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash M : B}$$

The methodology:

1. You have a corresp to formal logic(s) $\Gamma \vdash A$ props $\rightsquigarrow$ $\Gamma \vdash A$ type, often by construction. But then you have strange types like nat and list that don't really corresp to props.

$\Gamma \vdash A$ true $\rightsquigarrow$ $\Gamma \vdash M : A$ $(\exists M)$.

2. It ought to be constructive in two senses (?):

a) the proof terms have a computational ~~is~~ & interpretation and corresp to programs. Or so you would like: many fail at this, like HoTT.

Computational TT

Type theory is all about computation. Type theory is computational foundations in that it is a self-standing theory of truth. i.e., Brouwer: truth is based on computation because that is the fundamental human faculty. Start with a programming language: a deterministic transition system $M \mapsto M'$ with a specification of when you're finished: M Value/canonical $M \Downarrow V$. Then you define what are types and elements using meaning explanations in terms of computation.

1. Types are certain programs: they run, and when evaluated, their values designate a specification E.g. nat specifies programs that evaluate either to zero or the successor of another. Then "if true then nat else Π" is a type!

2. Elements of a type are programs that satisfy or adhere to the spec given by the type (in terms of eval) E.g.: if true then ~~true~~ 0 else false $\in$ nat,

$\in$ if true then nat else Π.

$\Gamma \gg M \in A$ } judgments of CTT.
$\Gamma \gg A$ type }

There are in two senses checkable "check equality"

$\Gamma \gg M \doteq N \in A$ }
$\Gamma \gg A \doteq B$ }

b) absence of LEM or DNE or... This is desired for "axiomatic freedom": the fewer axiomatic commitments you have, the freer you are to explore in diff directions. Somehow it's the computational grounds that get rid of LEM.

RWH thinks a) is the engine and b) is the smoke.

3. Decidability. The judgments are computably enumerable by definition and people insist on decidability: $\Gamma \vdash M : A$ or not

$\Gamma \vdash A$ type or not

$\Gamma \vdash M = N : A$ or not

$\Gamma \vdash A = B$ or not.

idea: Type checking is proof checking. (The complexity of the decision problem — "horrendous beyond belief" — is conveniently ignored)

4. Computational meaning, if any, is given extrinsically, imposed after the fact.

↳ What they're really interested in is the interpretation of their formal system into various mathematical structures, à la model theory.

eg. abs $\doteq$ id $\in$ nat $\to$ nat

abs $\neq$ id $\in$ int $\to$ int

Specify behaviour instead of structure.

The role of proof theories is to access the truth. Not to state definitive definitions of anything. The proof theories will be inductively defined

$\Gamma >> A$ [Lept M]

↳ "A is true" extracts M.

This is an example of a refinement logic (Constable and Bates)

(Chet Murthy ported the ideas of NuPRL to Coq during a workshop at INRIA).

Don't care about the proof theory's properties: only that it's useful.

2018-01-18

Case Study: The Simply Typed λ-calculus $\mathcal{T}$.
functions + base types.

types $A ::= b \mid A \to B$ ← base type, for illustr today w/ 2 distinct inhabs
terms $M ::= x \mid c \mid \lambda x{:}A.\, M \mid MN$
ctxts $\Gamma ::= \cdot \mid \Gamma, x{:}A$ distinct variables

$\Gamma \vdash M : A$ is the typing judgment
$\Gamma \vdash M \equiv N : A$ is definitional equality } ind defined.

In particular, closed terms $M : A$ (perhaps base type b)

Technical stuff: $[M/x]M'$ is substitution
$\gamma : \Delta \to \Gamma$, $\Delta \vdash \underbrace{\gamma(x)}_{\text{term}} : A$ for each $x : A \in \Gamma$
$\hat{\gamma}(M)$ extension to terms.

Any notion of computation comes from the outside and is extrinsic.
$\Gamma \vdash M \equiv N : A$ includes computation and is Refl Sym Trans Congruence
For example:

$$\frac{\Gamma, x{:}A \vdash M : B \qquad \Gamma \vdash N : A}{\Gamma \vdash (\lambda x{:}A.\, M) \equiv [N/x]M : B}.$$

Computation: We'll look at two computational interpretations of this formalism:
1. closed terms aka programs (maybe of base type). This greatly influences the meaning of variables and equality. Closed evaluation ↳ range over closed terms (values)
2. open terms of any type and open reduction. Variables range over open terms.

Intuition:
If $x{:}A \vdash M : B$, x is restricted to closed terms, ~~equality~~ much fewer things are going to be equal. E.g.
$x{:}\mathrm{nat} \vdash M \equiv N : \mathrm{nat}$
only for M, N constant natural numbers.

Over open terms, then you can get
$x:nat \vdash x+x \equiv 2x : nat$.

1. Closed computation

We'll define 3 judgments:

$M \text{ val}$ $\quad M \Downarrow V \triangleq M \longmapsto^* V \text{ val}$

$M \longmapsto M'$.

$$\frac{}{c \text{ val}} \qquad \frac{}{\lambda x{:}A.M \text{ val}} \qquad \frac{M \longmapsto M'}{MN \longmapsto M'N}$$

"head reduction" (though this term is usually reserved for open reduction ~~⊖~~ (Something also typically used in proofs?)

$$(\lambda x{:}A.M)N \longmapsto [N/x]M$$

Evaluation contexts

$E ::= \bullet \mid EM$ "spine"

Define $E\{M\}$ to fill the hole $\bullet$ in M.
Don't need to worry much because we're dealing only w/ closed terms.

We can now write

$M \longmapsto M'$ iff $M = E\{P\}$, $M' = E\{P'\}$, and $P \xrightarrow{\text{contr}} P'$ (Evaluation Context Semantics (Felleisen))

$(\lambda x{:}A.M)N \xrightarrow{\text{contr}} [N/x]M$

Felleisen thinks ECS is easier to formalize
Bob thinks SOS is easier to formalize and is what he and Karl used to formalize SML in Twelf.

Goal: Show that all well-typed programs terminate. (closed)

(X) Claim: If $M : A$ then $M \Downarrow$ $(\exists V.\ M \Downarrow V)$

We will then want to show:

Goal: Show that all well-typed open terms normalize:
If $\Gamma \vdash M : A$ then M norm$_\beta$.

Pf of (*): By ind on syntax of M.

Var: impossible

Const $M = c$: Then $A = b$ and c val V

λ: $M = \lambda x:B.N$ $A = B \to C$, λ is a val V

App: $M = NP$, $N : B \to A$ $N \Downarrow$
$P : B$ $P \Downarrow$
(ind hyp)
TS: $NP \Downarrow$. Not obvious.

Notice: $(\lambda x.xx)(\lambda x.xx)$ doesn't terminate.

So we're stuck: application of terminating terms need not terminate. But we're saved by the fact that we have types!

Idea: Strengthen the IH! (Idea due to Tait?)

Define a property called hereditary termination $HT_A(M)$ s.t.
1. implies termination
2. induction goes through.

Defined by ind on A.

$HT_b(M)$ iff $M \Downarrow$ behavioural, not structural cond.
$HT_{A \to B}(M)$ iff $\forall N$, ($HT_A(N)$ implies $HT_B(MN)$)

This is called a "negative" formulation. There exists a positive formulation.

For fun, imagine we had polymorphism. $HT_{\forall \alpha:A}(M)$?

Claim: If $M:A$ then $HT_A(M)$ ☠

Pf: Var x impossible

const $c:b$ then $HT_b(c)$ ✓

app $MN:B$ b/c $M:A\to B$, $N:A$

$HT_{A\to B}(M)$ $HT_A(N)$

$\therefore HT_B(MN)$ ✓

lam: $\lambda x:A.M : A\to B$ b/c $x:A \vdash M:B$

Stuck! can't apply IH b/c M is not closed!

TS: $HT_{A\to B}(\lambda x:A.M)$.

Suffices TS:

if $HT_A(N)$ then $HT_B((\lambda x:A.M)N)$.

head reduct!

Suppose $HT_A(N)$. Know that

$(\lambda x:A.M)N \longmapsto [N/x]M$.

Idea: Say something about open terms as "mappings" given by subst.

$x:nat \vdash x+1 : nat$

"$nat \longrightarrow nat$".

Strengthen the IH even further:

If $\Gamma \vdash M:A$ then

for all $\gamma : \cdot \longrightarrow \Gamma$ (closing subst)

if $HT_\Gamma(\gamma)$, then $HT_A(\hat{\gamma}(M))$.

Cor: If $M:b$ then $M\downarrow$

Pf: Var: $\Gamma', x:A \vdash x:A$, $\hat{\gamma}(x) = \gamma(x) : A$.

(Γ)

Our assumption $HT_A(\gamma(x))$ ✓

Const: immediate $\quad \boxed{\hat{M} := \hat{\gamma}(M)}$

App: By IH, $HT_{A\to B}(\hat{M})$, $HT_A(\hat{N})$

$\therefore HT_B(\hat{M}\hat{N})$, and $\widehat{MN} = \hat{M}\hat{N}$.

17

lam $\dfrac{\Gamma, x:A \vdash M : B}{\Gamma \vdash \lambda x:A.M : A \to B}$.

Suppose $HT_\Gamma(\gamma)$. TS: $HT_{A\to B}(\lambda x:A.\hat{M})$.

Suppose $HT_A(N)$. TS: $HT_B((\lambda x:A.\hat{M})N)$.

By IH, $HT_B([N/x]\hat{M})$ because

$HT_{\underbrace{\Gamma, x:A}_{\Gamma'}}(\underbrace{\gamma[x \mapsto N]}_{\gamma'})$, i.e., $HT_B(\hat{\gamma'}(M))$

[Lemma ("head expansion"). If $HT_A(M')$ and $M \mapsto M'$, then $HT_A(M)$.
↳ determinism matters.]

The result then follows by head expansion and
$(\lambda x:A.\hat{M})N \mapsto [N/x]\hat{M}$. ∎

2018-01-23

Last time

We tried showing

Thm: If $M : A$, then M term$_\beta$.

We introduced hereditary termination to accomplish this. We defined:

- $HT_b(M)$ iff M term$_\beta$
- $HT_{A\to B}(M)$ iff $\forall N$ if $HT_A(N)$ then $HT_B(MN)$

Instead of showing M term$_\beta$, we show $HT_B(M)$ and then

Thm: If $HT_A(M)$ then M term$_\beta$.

We had to generalize to

Thm: If $\Gamma \vdash M : A$ and $HT_\rho(\gamma)$, then $HT_A(\hat{\gamma}(M))$.

Here

$HT_\rho(\gamma)$ iff $\forall x : A \in \Gamma$, $HT_A(\gamma(x))$, and $\mathrm{dom}\,\Gamma = \mathrm{dom}\,\gamma$.

Remark: $HT_\emptyset(\phi)$, $\hat{\phi}(M) = M$.

Lemma (head expansion):
If $HT_A(M')$ and $M \longmapsto M'$ then $HT_A(M)$.

Pf: By induction on the structure of A.

Case $A = A_1 \to A_2$:

Have $HT_{A_1\to A_2}(M')$ and $M \longmapsto M'$.

TS: $HT_{A_1\to A_2}(M)$.

Suppose $HT_{A_1}(N)$. STS $HT_{A_2}(MN)$.

By assumption $HT_{A_2}(M'N)$ so by ind (on the type!), $HT_{A_2}(MN)$. □

This works exactly because we have head reduction $\dfrac{M \longmapsto M'}{MN \longmapsto M'N}$. This is why head reduction is the way it is.

(+) Thm: If $HT_A(M)$, then M term$_\beta$.

Pf: By induction on A.

Case. $A = b$. immediate

Case $A = A_1 \to A_2$: Suppose $HT_A(M)$. Then we're stuck, because we don't know where to get an argument of type A_1.

What to do?

1. Give up! Expect properties only at observables / base type. Change the theorem to: If $M : b$, then M term$_\beta$

2. Change the definition of $HT_{A_1 \to A_2}(M)$ to also insist M term$_\beta$. Then (+) becomes true by definition. But you need to push this requirement through everywhere else. But this is easy because we know termination is closed under head expansion.

Another way — the positive (or the "non-negative") way — of defining HT:

- $HT_b(M)$ iff $M \longmapsto^* c$ $(\exists c : b)$ (no real change)
- $HT_{A_1 \to A_2}(M)$ iff $M \longmapsto^* \lambda x{:}A.M_2$ and (*) for all M_1 s.t. $HT_{A_1}(M_1)$, we have $HT_{A_2}([M_1/x]M_2)$.

This is called the method of canonical forms:

$HT_A(M)$ iff $M \longmapsto^* V$ val s.t. $\exists$ a $HV_A(V)$. $CF_A(V)$.

Where $HV_b(c)$ s.t. $c : b$ and $HV_{A_1 \to A_2}(\lambda x{:}A.M)$ s.t. $\forall$ (*). (+)

Then the intro rules become easy:

$$\frac{\Gamma, x:A_1 \vdash M_2 : A_2}{\Gamma \vdash \lambda x:A_1.\, M_2 : A_1 \to A_2} \Big\}^{\forall M_1 \text{ s.t. } HT_{A_1}(M_1)} HT_{A_2}([M_1/x]M_2).$$

OTOH, elim rule is not immediate and you need head expansion. □

(For comparison you can write

$HT^+_A(M)$ & $HT^-_A(M)$.)

Let's go back to the negative form.

Claim: The "give up" option only works for CBN! You only ever observe termination at base type under CBN, not CBV. Why?

The key idea of CBN is that you evaluate something of function type only if it is applied.

CBV: Not true! $F(M)$ $(A \to B) \to C$ $A \to B$ You need to evaluate M regardless of whether you evaluate it.

Pos

$HV^{cbn}_{A_1 \to A_2}(\lambda x:A.\, M)$ iff $HT^+_{A_1}(M_1) \supset HT^+_{A_2}([M_1/x]M)$

$HV^{cbv}_{A_1 \to A_2}(\lambda x:A.\, M)$ iff $HV_{A_1}(V_1) \supset HT^+_{A_2}([V_1/x]M)$.

Now consider what happens when you try to show (cbv)

Thm: If $\Gamma \vdash M:A$ and $HV_\Gamma(\gamma)$, then $HT^+_A(\hat{\gamma}(M))$.

Then $HT^+_{A \to A_2}(M)$ and $HT^-_{A_1}(N)$. IS: $HT^+_{A_2}(N)$.

But you must prove

$N \longmapsto^* V$ and $HV_A(V)$. But here you can't just give up: you must show this.

To summarize:

i) If $\Gamma \vdash M : A$ and if $HT_\rho(\gamma)$, then $HT_\rho(\hat{\gamma}(M))$.
The formalism/syntax "$\Gamma \vdash M : A$" has a semantics "if $HT_\rho(\gamma)$, then $HT_\rho(\hat{\gamma}(M))$."

Notation: ii) A type (grammar)
ii) $M \in A$ iff $HT_A(M)$
↳ this is a behavioural specification. $M:A$ has no such meaning: you just derived it from a bunch of rules. You can, however, prove the implication we did!
iii) $\Gamma \gg M \in A$ means $\gamma \in \Gamma$ implies $\hat{\gamma}(M) \in A$.

The fundamental theorem then says:

Safety { If $\underbrace{\Gamma \vdash M : A}_{\text{formally provable, derivable}}$, then $\underbrace{\Gamma \gg M : A}_{\text{true}}$.

grammatical things make true statements about their behaviour.

What you care about is the truth and a formalism acts simply as a window on the truth and a means of accessing the truth, and as such is never canonical.

Aside: the type annotations are irrelevant and are only there for protocol.

$|M|$ means: $|\lambda x{:}A.M| = \lambda x.|M|$

Changing gears:

Normalization

Interpret $\Gamma \vdash M : A$ as a mapping on open terms (not closed / values)
→ variables are indeterminates.

Now the notion of behaviour is given by

M norm$_\beta$ means $M \longrightarrow_\beta^* N$ nf$_\beta$.

$(\lambda x:A.M)N$ cont$_\beta$ $[N/x]M$.

$$\frac{M \text{ cont}_\beta N}{M \longrightarrow_\beta N} \qquad \frac{M \longrightarrow_\beta M'}{MN \longrightarrow_\beta M'N} \qquad \frac{N \longrightarrow_\beta N'}{MN \longrightarrow_\beta MN'}$$

$$\frac{M \longrightarrow_\beta M'}{\lambda x:A.M \longrightarrow \lambda x:A.M'} \quad (!)$$

Reduction is ~~restricted to~~ defined for open terms.

N nf$_\beta$ iff $N \not\longrightarrow_\beta$.

Ex: ind characterize N nf$_\beta$.

Thm: If $\Gamma \vdash M : A$, then M norm$_\beta$, even for open terms!

Pf: By induction on typing.

We again get stuck at application.

M norm$_\beta$ N norm$_\beta$

TS: MN norm$_\beta$?

Stuck ☹ again.

Def (wrong / inadequate): ~~Tait~~ (Tarski norm)

$HN_b(M)$ iff M norm$_\beta$

$HN_{A_1 \to A_2}(M)$ iff if $HN_{A_1}(M_1)$ then $HN_{A_2}(MM_1)$.

Idea:

1) If $\Gamma \vdash M : A$, then ~~$HN_A(M)$~~
if $HN_\rho(\gamma)$, then $HN_A(\hat{\gamma}(M))$.

treat as a mapping:

Problem: Want to interpret the image of vars as also being open. So write $\Delta \vdash \gamma : \Gamma$ for $\gamma : \Delta \to \Gamma$, then:

→ if $HN^\Delta_\rho(\gamma)$, then $HN^\Delta_A(\hat{\gamma}(M))$.

2) $HN^\Delta_A(M)$ implies M norm$_\beta$

3) $HN^\Gamma_\rho(id_\Gamma)$.

2017-8-25

Terminology / credits:

• hereditarily X	(most common)
• logical relations	(Statman)
• Tait's method	(W. Tait)
• (Girard's method)	(J-Y Girard)
• computability method	(Tait)
• reducibility method	(Girard ?)

Recall that we generalized our account to cope with open terms and normalization:

$$HN^\Delta_A(M) \quad \text{s.t.} \quad \Delta \vdash M : A$$

We tried

Thm: $\Gamma \vdash M : A$ implies M norm$_\beta$ ✗ Church

↓ then try

Lemma: $\Gamma \vdash M : A$ implies $HN_A^\Gamma(M)$ ✗ prove these directly

↓ then try

(♯) Lemma: If $\Gamma \vdash M : A$ and $\Delta \vdash \gamma : \Gamma$ and $HN_B^\Delta(\gamma)$ then $HN_A^\Delta(\hat{\gamma}(M))$.

Our first attempt at $HN_A^\Delta(M)$ will be too naive:

$HN_A^\Delta(M)$ def. $\Delta \vdash M : A$

$HN_b^\Delta(M)$ iff M norm$_\beta$ [$HN_{A_1}^\Delta(M_1)$ implies $HN_{A_2}^\Delta(MM_1)$]

$HN_{A_1 \to A_2}^\Delta(M)$ iff

Lemma

1. $HN_\Gamma^\Gamma(id_\Gamma)$ — i.e., $HN_A^{\Gamma, x:A}(x)$ ($x_1 := x_1, \ldots, x_n := x_n$)
2. $HN_A^\Gamma(M)$ implies M norm$_\beta$.

Pf of (♯):

- $\Gamma, x:A \vdash x : A$. Then $\hat{\gamma}(x) = \gamma(x)$ $HN_B^\Delta(\gamma(x))$ ✓ by assumption
- $\Gamma \vdash c : b$, $\hat{\gamma}(c) = c$ norm$_\beta$. ✓
- $\dfrac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M(N) : B}$ $\left.\begin{matrix} HN_{A\to B}^\Delta(\hat{M}) \\ HN_A^\Delta(\hat{N}) \end{matrix}\right\}$ IH

$\therefore HN_B^\Delta(\hat{M}\hat{N})$ ✓ ($\hat{M}\hat{N} = \widehat{MN}$)

comes from calculus

• $\Gamma, x:A \vdash M:B$ IH: If $HN_A^\Delta(N)$, then $HN_B^\Delta([N/x]\hat{M})$

$\Gamma \vdash \lambda x:A.M : A \to B$ ↕ head reduces

TS: $HN_{A\to B}^\Delta(\lambda x:A.\hat{M})$.

Suppose: $HN_A^\Delta(N)$. TS: $HN_B^\Delta((\lambda x:A.\hat{M})N)$

Result follows by head expansion. D.

So still need to show:

Lemma:

3. If $M \mapsto M'$ and $HN_A^\Delta(M')$ then $HN_A^\Delta(M)$.

Pf: (3) Proc by ind on A.

• $A=b$: immediate, because $M \mapsto M'$ norm$_\beta$ implies M' norm$_\beta$.

• $A = A_1 \to A_2$: Suppose $HN_{A_1\to A_2}^\Delta(M')$ and $M \mapsto M'$.
WTS: $HN_{A_1\to A_2}^\Delta(M)$.

Suppose: $HN_{A_1}^\Delta(M_1)$. STS: $HN_{A_2}^\Delta(M(M_1))$

↓

$HN_{A_2}^\Delta(M'(M_1))$.

But that follows by the IH, because the type got smaller. D

Pf of (1) and (2): By ind on A.

1. • $A=b$: TS: $HN^{\Gamma, x:b}(x)$. i.e., x norm$_\beta$ ✓

• $A = A_1 \to A_2$. TS: $HN^{\Gamma, x:A_1\to A_2}(x)$.

Spse $HN_{A_1}^{\Gamma,x:A_1}(M_1)$ TS: $HN_{A_2}^{\Gamma,x:A}(x(M_1))$.

We're a bit stuck: x applied to something... But!

By IH(2), M_1 norm$_\beta$. But we're stuck, so we need to strengthen the IH. See (17).

2. $A = b$: spse $HN_b^\Gamma(b)$. TS: M norm$_\beta$. ✓

$A = A_1 \to A_2$. spse $HN^\Gamma_{A_1 \to A_2}(M)$. TS: M norm$_\beta$.

But we don't have any A_1 to apply this to, so we're stuck.

But IH_1, (a) $HN^\Gamma_{A_1}(x)$ (dubious step, but ignore this)

$\therefore HN^\Gamma_{A_2}(Mx)$. $\therefore$ by IH_2, Mx norm$_\beta$

(E) $\therefore M$ norm$_\beta$.

check this inference carefully.

↳ Dubious because Γ might be empty or not have any variables of type A_1.

The standard trick for decades was to use "saturated contexts" Γ_∞ containing ∞-many vars for ∞-many types, and then you show a completeness property where if $\Gamma_\infty \vdash M : A \Rightarrow$ finite Γ. $\Gamma \vdash M : B$. (ctd on 18)

Strengthen (the old way) slightly clumsy. (See (9) for cleaner form)

1. If $\Gamma \vdash x : A_1 \to A_2 \to \cdots \to A_k \to A$ and M_1 norm$_\beta$ $\cdots$ M_k norm$_\beta$ then $HN^\Gamma_A(x M_1 \cdots M_k)$.

Patch up pf of 1.

$A = b$: TS: $HN_b^\Gamma(x M_1 \cdots M_k)$. B/c M_i norm$_\beta$, M_i normalizes to N_i,

so $x M_1 \cdots M_k \to_\beta^* x N_1 \cdots N_k \not\to_\beta$.

(Just normalize the M_i to normalize $x M_1 \cdots M_k$).

$A = A_1 \to A_2$: TS: $HN^\Gamma_{A_1 \to A_2}(x M_1 \cdots M_k)$

spse $HN^\Gamma_{A_1}(M_{k+1})$. TS: $HN^{\Gamma, x:A}_{A_2}(x M_1 \cdots M_k M_{k+1})$

By $IH(2)$ M_{k+1} norm$_\beta$. So by $IH(1)$ ↗

(17)

A better approach than $\vdash_\Sigma$:

Idea: Kripke semantics and presheaves.

With our computer hacker hat on, you realize that what you really want to do is allocate new variables for (+) You want malloc.

Worry: Why should the property of hereditary normalization be stable under such an act. I've proven these facts in a world w 100 variables, how do I know it will still hold in a world w 101 variables?

Key: Only consider facts that will be true in all future worlds.

So we change the definition of HN:

Let $\Delta' \geq \Delta$ if Δ' extends Δ ("future world"). (trans & refl)

~~If $\Delta' \geq \Delta$ and~~ Then

$HN^{\Delta}_{A_1 \to A_2}(M)$ iff if $\Delta' \geq \Delta$ and $HN^{\Delta'}_{A_1}(M_1)$ then $HN^{\Delta'}_{A_2}(M M_1)$ } demand that functs anticipate alloc of new vars.

Fact (monotonicity / functoriality):

If $HN^{\Delta}_{A}(M)$ and $\Delta' \geq \Delta$, then $HN^{\Delta'}_{A}(M)$.

Now you can go through the proof and allocate as you go. At (+), "by IH$_1$, $HN^{\Gamma, x:A_1}_{A_1}(x)$".

Lem: If $\Gamma \vdash M : A$ and $\Delta \vdash \gamma : \Gamma$ and $HN^{\Gamma}_{A}(M)$, then $HN^{\Delta}_{A}(\hat{\gamma}(M))$

Rmk: $\Delta, x:A \vdash id_\Delta : \Delta$, i.e., weakening is a substitution.

Go through all the proofs to push through the above. They should go through straightforwardly.

Lemmas: $\mathcal{E} ::= \cdot \mid \mathcal{E}(M)$

1) If $\mathcal{E} : \Gamma \Rightarrow A$ — Don't you need some context here?

$\mathcal{E}$ hole has type C, whole thing has type A.

and $\mathcal{E}$ norm$_\beta$ (all arguments M are norm$_\beta$)

then $HN^{\Gamma, x:C}_{A}(\mathcal{E}\{x\})$. (Cleaner statement of (17))

Use, $\mathcal{E}\{x\}(M) = \mathcal{E}(M)\{x\}$.

2) if $HN^{\Delta}_{A}(M)$ then M norm$_\beta$.

Rant about strong normalization.

There is an obsession in the literature about strong ~~so~~ normalization, SN$_\beta$.

RWH claims that the only important thing about SN$_\beta$ is

→ Never important in itself!

Only useful as a means to other things, as a tool.

Usual definition:

no ∞ reduction sequences:

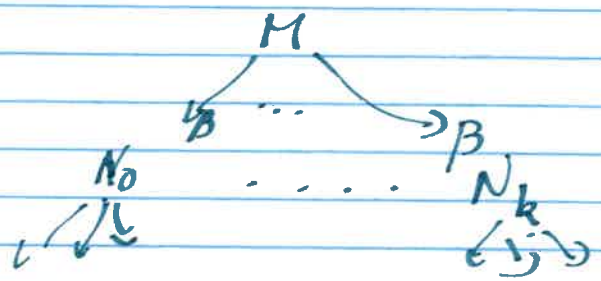
Already suspect as a construction it $\neg \exists M.\ M = M_0 \xrightarrow[\beta]{} M_1 \xrightarrow[\beta]{} M_2 \xrightarrow[\beta]{} \cdots$

What this means is that the principle of transfinite induction on reduction is valid:

$$(\forall M\,((\forall N\, \langle M \to N, P(N)\rangle \supset P M) \supset \forall M\, P(M)).$$

So no inf red sequences = TIR is valid.
This follows from König's lemma

Each M has finitely many β redexes $N_0, \ldots, N_k$



finite fanout, all branches are finite by SN, so tree finite by König.

This is the only ~~way~~ reason you care about SN.

By $E : C \hookrightarrow A$, (part (1) of the lemma on 2018-01-30. (19)),
TRUTH means
$\Delta \vdash x : C, \quad \Delta \vdash E : (C \hookrightarrow A)$)

[chtt-s18.slack.com]

Last time:

FTLR: If $\Gamma \vdash M : A$ and $\gamma : \Delta \to \Gamma$, then $HN^{\Delta}_{\Gamma}(\gamma)$ implies $HN^{\Delta}_{A}(\hat{\gamma}(M))$. ⊕

Cor: If $\Gamma \vdash M : A$, then M $norm_\beta$.

→ Structure "$\Gamma \vdash M : A$" determines behaviour "$HN^{\Delta}_{\Gamma}(\hat{\gamma}(M))$".

To show ⊕, you need to show that $HN^{\Gamma}_{\Gamma}(id)$, and to do so, you must show that

if $\Delta \vdash x : A$, then $HN^{\Delta}_{A}(x)$.

But you get this by using

if $\mathcal{E}$ $norm_\beta$, then $HA^{\Delta}_{A}(\mathcal{E}\{x\})$ ($\mathcal{E} : \mathbf{C} \rightsquigarrow A$, $\Delta \vdash x : C$).

Nts: $\lambda x.M$ is HN when M is HN.

Nts: $HN^{\Delta}_{A_1 \to A_2}(M)$ implies M $norm_\beta$.

Idea: Allocate a fresh variable at type A_1 and consider Mx at A_2.

∴ Mx $norm_\beta$

∴ M $norm_\beta$.

Claim: Kripke logical rel'ns are the means for talking about state in PL.

When allocating a fresh variable, you move from Δ to $\Delta, x : A$, where $x \notin \Delta$. But you need a stability property, which tells you that moving to a future world does not affect hereditary normalization.

(HN_A : Worlds → Rel (↪ Sets))

Δ ordered by ≤

Exercise: (closed and open — do both)

Add "negative" products: → defined by elim form / projections,

$\langle M_1, M_2 \rangle$ val

$M_A \cdot 1$, $M_A \cdot 2$ projections

$\langle M_1, M_2 \rangle \cdot 1 \longmapsto M_1$

$\cdot 2 \longmapsto M_2$.

Define 1) $HT_{A_1 \times A_2}(M)$ — reprove term.

2) $HN^{\Delta}_{A_1 \times A_2}(M)$ — reprove norm.

Positive Types

(Exercise: do positive products)

let $\langle\rangle$ be M in N

let $\langle x, y \rangle$ be M in $N_{x,y}$

} defined by pattern matching.

Prime example: booleans.

~~tt~~ true, false are values/intros/cforms.

$\text{if}_A(M; P; Q)$ elim

$\mathcal{E} ::= \cdots \mid \text{if}_A(\mathcal{E}; P; Q)$.

$\text{if}_A(\text{true}; P; Q) \longmapsto P$

$\text{if}_A(\text{false}; P; Q) \longmapsto Q$

$$\frac{}{\Gamma \vdash \text{true} : \text{bool}} \qquad \frac{}{\Gamma \vdash \text{false} : \text{bool}} \qquad \frac{\Gamma \vdash M : \text{bool} \quad \Gamma \vdash P : A \quad \Gamma \vdash Q : A}{\Gamma \vdash \text{if}_A(M; P; Q) : A}$$

Need to account for the fact that there exist infinitely many booleans in the open case, but that if only knows about two of them.

~~tt~~ if is a "recursor", which witnesses that bool is inductively defined by true, false.

Let's revisit the termination proof for closed terms. This should go through straightforwardly.

$HT_{bool}(M)$ iff either $M \longmapsto^* true$
or $M \longmapsto^* false$

Remark: for function types, you can give a pos or neg characterization. But with you're using positive types, you have no choice but to give a positive characterization of HT_{bool}.

Easy: If $HT_{bool}(M)$, then $M\ term_\beta$.

Easy: Head expansion.

FTLR: If $M \vdash M : A$ and $HT_\Gamma(\gamma)$, then $HT_A(\hat{\gamma}(M))$ } closed

Cor: If $M : A$, then $M\ term_\beta$.

Exercise: do HT for positive products.

$HT_2(M)$ iff $M \longmapsto^* \langle\rangle$

$HT_{A_1 \times A_2}(M)$ iff $M \longmapsto^* \langle M_1, M_2 \rangle$ s.t. $HT_{A_1}(M_1)$ and $HT_{A_2}(M_2)$.

Aside: Could I somehow give a negative characterization of bool, that is, defined in terms of if. If I could, I'd want to say something along the lines of

$HT_{bool}(M)$ iff
if { 1) $HT_A(P)$
2) $HT_A(Q)$
then $HT_A(if(M; P; Q))$. } What's wrong with this?

The problem with this that it violates definition by recursion on the type; ~~we define HTbool in terms~~

We're characterizing the behaviour of bool in terms of arbitrary types A.

Vertigo

$bool = \forall A.\ A \to A \to A$
are the Church booleans.
RWH claims they arose because of the above considerations.

End of Aside

Onto open terms

To show (norm. for open terms)

If $\Gamma \vdash M : A$ and $\gamma : \Delta \to \Gamma$, then $HN^{\Delta}_{\Gamma}(\gamma)$ implies $HN^{\Delta}_{A}(\hat{\gamma}(M))$.

How do we define $HN^{\Delta}_{bool}(M)$? (This won't generalize nicely to covers, e.g., nats)

Cannot: $M \mapsto^* true$ or $M \mapsto^* false$!
Because M is open and could be "x", or "xN" or "$if\ (xN; P; Q)$" or ...

Constraints: We must validate:

1) $HN^{\Delta}_{bool}(true)$ and $HN^{\Delta}_{bool}(false)$
2) if $HN^{\Delta}_{bool}(\hat{M})$ "(1)" and $HN^{\Delta}_{A}(\hat{P})$ "(2)" and $HN^{\Delta}_{A}(\hat{Q})$, (3) then $HN^{\Delta}_{A}(\hat{if}_{A}(\hat{M}; \hat{P}; \hat{Q}))$.

(May have to prove something stronger ...) → } motivated by structural rules.

0) $HN^{\Delta}_{bool}(M)$ implies M norm$_\beta$.

What do we know about $\hat{M}$? That $\hat{M}$ norm$_\beta$.

Fact: If M norm$_\beta$, then $M \mapsto^*_\beta N \not\mapsto_\beta$
You're forced to do the head-β-reductions. N norm$_\beta$.
$\hat{M}$ reduces to something that is head irred, i.e., in hnf.

Define $HN^{\Delta}_{bool}(M)$ iff M $norm_\beta$.

(This will only work for enumeration types, not natural numbers).

By def'n of (1), $\hat{M}$ $norm_\beta$.

~~By Lemma on (2) and (3), we have $\hat{P}, \hat{Q}$ $norm_\beta$.~~

Lemma (head exp) ...

Lemma (work horse).
1) If $HN^{\Delta}_{A}(M)$, then M $norm_\beta$
2) If E $norm_\beta$, then $HN^{\Delta}_{A}(E\{x\})$.

By Lemma on (2) and (3), we have $\hat{P}, \hat{Q}$ $norm_\beta$.

$\hat{M}$ $norm_\beta$, so $\exists N.\ N \not\mapsto$ and $M \rightarrow^* N$.
We proceed by case analysis on N.

N = true:
TS: $HN^{\Delta}_{A}(\text{if}(\hat{M}; \hat{P}; \hat{Q}))$
$\hat{M} \rightarrow$ true
$\hookrightarrow \hat{P}$, which we know $HN^{\Delta}_{A}(\hat{P})$ by (2).
So we're done by head exp.

N = false: symmetric.

Otherwise: What do we know?

N is in Inf and is of type bool. So it must be

This is the crucial part of Today's write-up: $N = E\{x\}$ s.t. E $norm_\beta$.

$\therefore HN^{\Delta}_{A}(E\{x\})$ by Lemma 2)
$\therefore HN^{\Delta}_{A}(\underbrace{\text{if}(E\{x\}; \hat{P}; \hat{Q})}_{E'\{x\}})$

then use head expansion.

2018-02-01

Today: Finish our discussion of positive types
Last time we discussed booleans:

"Running code": 1. Closed interp — variables range over ~~open~~ closed comps, compute by eval
"Dairy algebra": 2. open interp — variables are indeterminates, compute by calculations

~~Stta~~

Defined $HT_{bool}(M)$ iff $M \mapsto^* \{ \text{true}, \text{false} \}$: $M \Downarrow$

Δ is the "state" — available indeterminates

$HT^{\Delta}_{bool}(M)$ iff M $norm_{\beta}$ (1)

The important point is that there are infinitely many booleans in the open case and yet you can push through FTLR $\bar{\omega}$ (1).

Before bool, the logical order of ideas in the open case was:

depends: 1. head exp
2. FTLR
3. Workhorse lemma

Workhorse lemma invented by Bob this week

After bool, the order changes:

1. head exp
2. Workhorse lemma — needed to prove
3. FTLR

Let's look at

Sums more generally

Want the types
0 aka void
$A+B$

If you have 1, unit, $A+B$, then bool $= 2 \doteq 1+1$

RWH regrets calling nullary case "abort" — PFPL because students think it will cause abort.

Formal setting:

$$\frac{\Gamma \vdash M : \text{void}}{\Gamma \vdash \text{case}_A\, M \{\} : A}$$

$$\frac{\Gamma \vdash M : A+B \qquad \Gamma, x : A \vdash P : C \qquad \Gamma, x : B \vdash Q : C}{\Gamma \vdash \text{case}_C\, M \{1 \cdot x \hookrightarrow P \mid 2 \cdot x \hookrightarrow Q\} : C} \; (*)$$

$1 \cdot M = in[1]\{A,B\}(M)$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash 1 \cdot M : A+B} \qquad \frac{\Gamma \vdash M : B}{\Gamma \vdash 2 \cdot M : A+B}$$

+ dyn.

1. Closed setting: straightforward

$HT_{valid}(M)$ iff (Coloc)
i.e., $\rightarrow HT_{Void}(M)$ (any M)

pos conds. $\begin{cases} HT_{A+B}(M) \text{ iff either } M \longmapsto^{a} 1 \cdot N \text{ and } HT_A(N) \\ \quad \text{or } M \longmapsto^{a} 2 \cdot N \text{ and } HT_B(N). \end{cases}$

Check head exp, FTLR.

If you were to attempt a negative def'n, you'd run into the same problem as bool:

" $\forall X.\ (A \to X) \to (B \to X) \to X$ " (†)

The whole inductive character of our enterprise breaks down. Rmk: (†) comes from considering the negative forms and this is how you cope w/ these encodings in general.

2. Open case:
We'll more or less recapitulate last time

$HTV^{\Delta}_{A+B}(M)$ iff ?

Considering (⊛) will tell us what we need. By IH, we'll know:
1) $HTV^{\Delta}_{A+B}(M)$
2) if $HN^{\Delta}_{A}(N)$ then $HN^{\Delta}_{C}([N/x]P)$
3) if $HN^{\Delta}_{B}(N)$ then $HN^{\Delta}_{C}([N/x]Q)$

Semantics of judgment is by substituting Head normalizing terms $\Gamma, x:A \vdash P:C$

→ These are wrong, b/c they do not respect the fundamental property of maps, namely, that they be invariant under allocation.

So insert

So switch

2) if $\forall \Delta' \geq \Delta$, $HN_A^{\Delta'}(N)$ then $HN_C^{\Delta'}([N/x]P)$
3) ———— B ———— Q

TS: $HN_C^{\Delta'}(\widehat{case})$.

Think of how the proof may go:
~~Proceed by case analysis on~~

1) $\hat{M} \mapsto M'$: head reduction / head exp.

2) $\hat{M} \not\mapsto$, e.g., b/c $\hat{M}$ is a var

3)

Proceed by case analysis on $\hat{M} \mapsto^* NH$:

~~So: H~~

1) $N = 1 \cdot N'$: $\widehat{case} \mapsto [N'/x]\hat{P}$, and we want this to be $HN_C^{\Delta'}$. But then we want N' to be $HN_C^{\Delta'}$. And then you get what you want by head exp

2) $N = 2 \cdot N'$: symmetric.

3) $N = \mathcal{E}\{z\}$: Then
case $\mathcal{E}\{z\}$ $\{1 \cdot x \hookrightarrow P \mid 2 \cdot x \hookrightarrow Q\}$
Set $\mathcal{E}' = \underline{case}\ \mathcal{E}\ \{$ ——— $\}$
Then apply the workhorse lemma and done.

You can do this case analysis if $\hat{M}$ norm$_p$, b/c then it has a hnf. You can't have a weak divergence in a normalizing term.

No route to a normal form of $(\lambda x.N)PQ$ w/o going through $[P/x]N$.

$HN^{\Delta}_{A+B}(M)$ iff

1) M norm$_p$ and
2) if $M \mapsto^* 1 \cdot N$, then $HN_A^{\Delta}(N)$, and
3) if $M \mapsto^* 2 \cdot N$, then $HN_A^{\Delta}(N)$.

Rmk: Now we're in a perfect position to be able to do positive products
let <> be M in N
let <x,y> be M in N xy.

Natural Numbers : Gödel's T

$A ::= nat \mid A_1 \to A_2$
$M ::= zero \mid succ(M) \mid$
$\underline{rec}_A (P ; x.Q)(M)$
See PFPL for dynamic "primitive recursion".
N.B. Clash of terminology wrt primitive recursion. You can define Ackerman here, but Wikipedia will also tell you Ackerman is not PR.

Proof of normalization of T is equiv to proof of consistency of Peano A.

If you don't accept the proof, then it must be b/c you don't accept induction, so you don't accept formal arith, so you're a nihilist, and ~~...~~ RWH will shake hands w/ you & part ways.

$$\frac{\Gamma \vdash M : nat \quad \Gamma \vdash P : C \quad \Gamma, x : C \vdash Q : C}{\Gamma \vdash \underline{rec}_C (P, x.Q)(M) : C}$$

"$\forall X. X \to (X \to X) \to X$".

1. If $M : A$, then M $term_\beta$ (if you want, you can restrict A to "ans" of answer Values)
2. If $\Gamma \vdash M : A$, then M $norm_\beta$.

We're going to show a stronger prop than HN:

If $\Gamma \vdash M : A$ and $HT_\Gamma(\gamma)$, then $HT_A(\hat{\gamma}(M))$.

$HT_{nat}(M)$ iff ? This should imply M $term_\beta$.

We're talking about positive types, so you do the obvious thing:

$HT_{nat}(M)$ iff either $M \longmapsto^* zero$
or $M \longmapsto^* succ(N)$ and $HT_{nat}(N)$.

Remark: The definition is circular and not all circular defns make sense, so you need to be careful.

You have "vertical" HT inductions defined on structure of A.

This is a "horizontal" one at the level of nat.

Define $HT_{nat}(-)$ to be the strongest P on nat s.t. if $M \longmapsto^* zero$, then $P(M)$
if $M \longmapsto^* succ(N)$ and $P(N)$, then $P(M)$

(≠) wrong ind

Alternatively, you could say

$$HT_{nat} = \bigcup_{i \geq 0} F^{(i)}(\emptyset),$$

Where $HT^{(0)}_{nat}$ never, and $HT^{(k+1)}_{nat}(M)$ is defined in terms of $HT^{(k)}_{nat}$ ...

At the end of the day, you need to accept some version of the above. It's a psychological matter, and depending on how you've been "conned", you'll find a diff formulation acceptable.

You can't say you believe in set theory but not nats!

In the rule (*) you're going to need to curry the M out and write some abuse of the form. reformulation

$$\frac{\Gamma \vdash P : C \qquad \Gamma, x : C \vdash Q : C}{P, \eta : nat \vdash rec(P; x\,Q)(\eta) : C}$$

The point is that that case is proven by the horiz induction principle.

~~and you're applying it to some~~

The P of (≟) is

$$P(N) \text{ if } HT_c(\widetilde{rec}(N)).$$

Rmk: (≟) is not mathematical induction! You're reasoning about a program's computation!

If you accept (≟), then you're accepting the consistency of PA.

Part 2: Do it all over again for open terms.

~~If~~ $HN^{\Delta}_{nor}(M)$ if M $norm_B$ ~~and~~

and 1) if $M \to^* zero$ then (true).
2) if $M \to^* succ(N)$, then $HN^{\Delta}_{nor}(N)$.

The story so far:

- products
- sums
- functions
- inductive types (coinductive types)

We understand these as
1) computation (CS / PL)
2) calculation (algebra).

"You want to be with the Nick Benton's of the world Obviously one of my favourite computer scientists"
— RWH

Next

1. Equality!
2. Richer type structure
   a. polymorphism
   b. dependent types
   c. cubical type theory.
   :

} need to talk about equality before you can talk about these.

Let's talk about:

Equality

Formal Type Theory

"definitional equality" (forget that definitions have any meaning)

Introduce a judgment

$$\boxed{\Gamma \vdash M \equiv M' : A}$$

Inductively def'd by rules, i.e., recursive. ⇒ worldly inhabitants

$\equiv$ is refl, trans, symm, congruence, β-principles, ...

All of the issues of dep T.T can be found in disputes over $\equiv$.

This is where you encode computation in F-TT, except that it's not directed!

$$\frac{\Gamma \vdash M \equiv M' : A \to B \qquad \Gamma \vdash N \equiv N' : A}{\Gamma \vdash MN \equiv M'N' : B} \qquad \frac{\Gamma, x:A \vdash M : B \qquad \Gamma \vdash N : A}{\Gamma \vdash (\lambda x:A.M)N \equiv [N/x]M : B}$$

You get them horrid issues like

$$\lambda x:\mathrm{nat}.\lambda y:\mathrm{nat}.\, x+y \neq \lambda x:\mathrm{nat}.\lambda y:\mathrm{nat}.\, y+x$$

(32)

$0 + x \equiv x$ but you may have defined $+$ by recursion on first vs 2nd

last $x + 0 \not\equiv x$

Computational TT

"exact equality" – terminology by Carlo A and RobH?

(A type)

$M \in A$

$\{ M \Downarrow V$
$\{ HV_A(V)$

$M \doteq M' \in A$ — wildly not r.e., not even arithmetic!

"M & M' are exactly equal as elems of type A".

Recall examples from first lecture.

Maps respect exact equality!

$x.A \gg P \doteq P' \in B$

means

if $M \doteq M' \in A$ then

$[M/x]P \doteq [M'/x]P' \in B.$ } functionality

Open terms are functions of their variables.

Then the fundamental theorem tells us

FTCR If $\Gamma \vdash M \equiv M' : A$, then $\Gamma \gg M \doteq M' \in A$.

2018-02-06

So far, we've considered

nat bool 0 $+$ 1 $\times$ $\to$

inductive, pos neg

(won't do coinductive types — neg).

Today: polymorphism: $\forall^2, \exists^2$ — second order quants

This is type/prop quantification

$\forall X.A$, $\exists X.A$

↖ variable types

We'll begin by a formal definition:

$A ::= X \mid \text{ans} \mid A_1 \to A_2 \mid \forall X.A$ (→ pos)

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

$$\frac{}{\Gamma \vdash \uparrow : \text{ans}} \qquad \frac{}{\Gamma \vdash \downarrow : \text{ans}}$$

$$\frac{\Gamma, x:A_1 \vdash M_2 : A_2}{\Gamma \vdash \lambda x:A_1.M_2 : A_1 \to A_2} \qquad \frac{\Gamma \vdash M : A_1 \to A_2 \quad \Gamma \vdash M_1 : A_1}{\Gamma \vdash M M_1 : A_2}$$

$$\frac{\Gamma, X \vdash M : A}{\Gamma \vdash \Lambda X.M : \forall X.A} \qquad \frac{\Gamma \vdash M : \forall X.A \quad \Gamma \vdash B \text{ type}}{\Gamma \vdash M[B] : [B/X]A}$$

TLC's a bit hokey to mix term variables and type variables, so it's not uncommon to split contexts:

$\Theta; \Gamma \vdash M : A$

type vars: Θ (finite set of X)

term vars: Γ (finite set of $x:A$)

Erasure So far we've written

$M : A$ implies M term.

But really, we mean

(≠) $M : A$ implies $|M|$ terminates

because the notion of computation doesn't care about types.

Similarly, we really mean

$\Gamma \vdash M : A$ and $HT_p(\gamma)$ imply $HT_{\#}(\tilde{\gamma}(|M|))$

Idea: the semantics (i.e. $HT_{\#}$) is defined on erased terms!

In our polymorphic setting, there are two possible approaches.

(I) ...

$|\lambda X.M|^{I} = |M|^{I}$ "full erasure" / "type assignment"

$|M[A]|^{I} = |M|^{I}$ common in the Italian school, and was used in ML, but has problems.

Unsoundness in ML polymorphism is due to turning value $|\lambda X.M|$ to non-val $|M|$

(II) ...

$|\lambda X.M|^{II} = \lambda\_.|M|^{I}$ — delays

$|M[A]|^{II} = |M|^{II}(\downarrow)$ — activates

Our goal is to show (≠). It doesn't really matter which erasure style you use, but Rw++ likes II so we'll do that! $|M| = |M|^{II}$

Rite of Passage for any PL person:

a) Try to prove (≠) and fail

b) Find your own proof — there are many.

What makes the proof difficult is that you need to think in 2nd order

logic, and all of math is done in 1st order math. And Gödel's thm tells us that you cannot do the proof in FOL.

Obvious attempt

Key: Generalize to $HT_A(M)$ and use Tait's method. So, something like:

$HT_{\alpha ns}(M)$ iff $M \to^* V$ or T

(neg) $HT_{A_1 \to A_2}(M)$ iff $HT_{A_1}(M_1) \to HT_{A_2}(M(M_1))$.

But what do you do $\overline{\omega}$ $\forall X A$? or X?

Perhaps as a first cut stick to closed types.

$HT_{\forall X.A}(M)$ iff $\left(\forall \text{ closed } B,\ HT_{[B/X]A}(M[B])\right)$ | "generics". ⇒ you don't know what you're doing.

Why does RWH dismiss this?

Rmk: you don't need HT_γ in this case.

Rmk: Be careful: this is supposed to be defined on closed terms, in which case you wouldn't write $M[B]$, but $M(B)$.

Does not work! The issue is that it is not inductive! $[B/X]A$ can be larger than $\forall X.A$! E.g.

$\forall X. X \to X \rightsquigarrow (\forall X. X \to X) \to (\forall X. X \to X)$,

Your next failed move: use a different measure of size!

"Never gonna work".

Metamathematical reasons make it so. But remark you can do something very

similar for dependent types and succeed!

Another idea: Try to "factor out" the substitution: use

"Δ" is more cautious than Θ

$HT_A(M)\ [\delta \# \Theta : \Theta]$

└ open A: $\Theta \vdash A$ type
Closing with $\delta : \Theta$

$HT_X(M)[\delta : \Theta]$ iff $HT_{\delta(X)}(M)$

⋮

$HT_{\forall X.A}(M)[\delta:\Theta]$ iff for all closed B,

$HT_A(M(\forall))[\delta[X \mapsto B]; \Theta, X]$

Still ill-defined!

Try: Define HT on the skeleton of the type!

$HT_A(M)\ [(\delta, \eta) : \Theta]$ — δ, η go together

idea: $\eta(X)$ is the interp of X as given by δ.

$HT_X(M)[\delta[X \mapsto A]; \eta[X \mapsto T] : \Theta]$ iff $T(M)$ (T holds on M)

η — predicate

η' — want $\eta'(X)(M)$

⋮

$HT_{\forall X.A}(M)$ iff for all closed B "standard interp of B"

$HT_A(M(\forall))[\delta[X \mapsto B], X \mapsto HT_B(-)[\delta, \eta]]$

Remark: This is only a refactoring of the previous def'n. Except for the technical formulation.

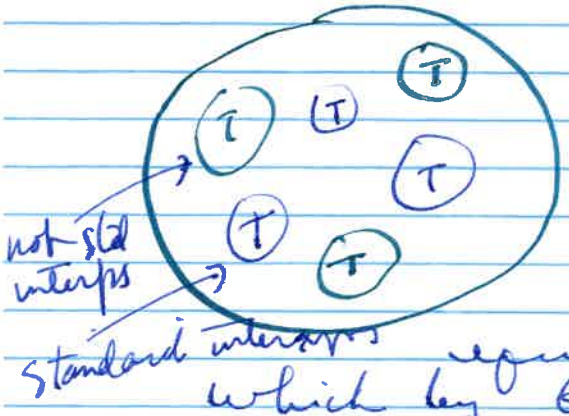
Rmk: The A is getting smaller!

This "standard interp" is what will hang you because you could have $B \cong A$ and you would be stuck.

What to do?

Type Candidates!

aka, admit non-standard interps of types.



"possible types".

You need to accept second-order comprehension — quantification over all types — to accept this proof. And the proof is equiv to consistency of FOArith., which by Gödel can't be done in FA. So you need something else, and SOA is this something else.

$HT_X(M)[\delta, \eta]$ iff $\eta(X)(M)$

$HT_{\forall XA}(M)[\delta, \eta]$ iff for all closed B — possible types T
$HT_A(M(\downarrow))[\delta[X \mapsto B]; \eta[X \mapsto T]]$.

Remark: We're not mentioning HT on the right! This is what saves us. HT is one of the meatballs in the soup, but we don't name it!

Obs: how can it be that you demand polymorphic types satisfy properties that aren't even expressible in the language?

What are type Candidates?
What made it possible to do the proof of the FTLR in the case of HT?

[What makes the proof work?]

- defn of "$HT_{A_1 \to A_2} = HT_{A_1} \to HT_{A_2}$"
  "$HT_{A_1 \times A_2} \quad HT_{A_1} \times HT_{A_2}$"
- Head expansion!
  - if $T(M')$ and $M \mapsto M'$, then $T(M)$.

So: possible type are sets of (closed ~~terms~~ erased terms) that are closed under head expansion.

2018-02-08

Girard's Method, aka Candidates

We want to define HT_A for systems F.
What to do about $HT_{A \to B}$?
Crucial idea: demand more of your programs than is evidently required, admitting NON-STANARD TYPES / TYPE CANDIDATES.
Use 2nd order comprehension — "sets of sets".
And allowing these inexpressible types is crucial for leveraging the full power of polymorphism, otherwise you end up with the impoverished notion of "generics".
They are not just a technical trick!

$HT_A(H)[\delta, \eta : \Theta]$ [η is a candidate assignment, the semantics of $\chi \in \Theta$]

good def'n

$HT_X(M)[\eta]$ iff $\eta(X)(M)$

$HT_{ans}(M)[\eta]$ iff $M, N \mapsto^* \downarrow$ or $\uparrow$

~~$HT_{A_1 \to A_2}(M)[\eta]$ iff~~

$HT_{A_1 \to A_2}(M)[\eta]$ iff $HT_{A_1}(M_1)[\eta]$ implies $HT_{A_2}(M(M_1))[\eta]$ (∓)

$HT_{\forall X.B}(M)[\eta]$ iff for all ~~closed~~ type candidates T, for all ~~closed~~ A, $HT_B(M[T])[\delta[X \mapsto A], \eta[X \mapsto T]]$. (∓)

Cannot restrict attention to standard types! Cannot insist $T = HT_A(-)[\delta, \eta]$.

What is a type candidate? collection of closed terms. It depends on what you want to prove.

1. Closed under head expansion: this is key to pushing the proof of HT through.
2. Depending on your goal, you may or may not need: if T is a type cand (t.c.) and $T(M)$, then M term.
3. If you had, e.g., recursion, then you need the admissibility of fixed point induction, see PFPL for details.

Remark: (∓) and (∓) are formulated negatively which will impact what you can get out of the theorem below. To get a props-as-types interp, you need a positive form, and you can push this through.

The FTLR becomes more technical:

FTLR

iff

Rmk: If you want "$HT_A(M)[\phi, \phi]$ implies M term$_B$" for all M, then you need to use the pos formulation of (∓), (∓). Otherwise, we restrict $A = ans$.

FTLR

key: interpret this as a map

If $\Theta \Gamma \vdash M : A$, and
$\delta : \Theta$ closed types
$\eta : \Theta$ candidate assignment
$HT_\Gamma(\gamma)[\delta, \eta]$.
(double-up idea)

then
$HT_A(\hat{\gamma}\,|\,\hat{\delta}(M)|)[\delta, \eta]$.

Rmk: $|\hat{\delta}(M)| = |M|$! So we use

$HT_A(\hat{\gamma}|M|)$

as our conclusion instead.

"There are those who teach, and those who use powerpoint" — RWH, on the use of chalk.

Lemma (Compositionality)
We want to relate $HT_{[A/X]B}(M)[\delta, \eta]$ and $HT_B(M)[\delta[X \mapsto A], \eta[X \mapsto HT_A(-)[\delta, \eta]]]$. Claim:
$HT_{[A/X]B}(M)[\delta, \eta]$ iff $HT_B(M)[\delta[X \mapsto A], \eta[X \mapsto HT_A(-)[\delta, \eta]]]$.

Pf: By induction on the structure of B. ∎

Lemma: For all A, δ, η, $HT_A(-)[\delta, \eta]$ is closed under head-exp. Pf: Is canned. ∎

Pf of FTLR:

Case 1) $\Theta, X; \Gamma \vdash M : B$ ∴ $\Theta; \Gamma \vdash \Lambda X.M : \forall X.B$

Fix $\delta : \Theta$, $\eta : \Theta$, $HT_\Gamma(\gamma)[\delta, \eta]$.

WTS: $HT_{\forall X.B}(\hat{\gamma}(|\lambda\_.|M||))[\delta, \eta]$.

Rmk: $\hat{\gamma}(|\lambda\_.|M||) = \lambda\_.\hat{\gamma}|M|$.

Fix a closed A, candidate T.

STS: $HT_B((\lambda\_.\hat{\gamma}|M|)())[\underbrace{\delta[X \mapsto A]}_{\delta'}, \underbrace{\eta[X \mapsto T]}_{\eta'}]$.

by Lemma: STS: $HT_B(\hat{\gamma}|M|)[\delta', \eta']$.

By the IH, $HT_B(\hat{\gamma}|M|)[\delta', \eta']$. ✓

2) $\dfrac{\Theta;\Gamma \vdash M : \forall X.B \quad \Theta \vdash A \text{ type}}{\Theta;\Gamma \vdash M[A] : [A/X]B}$

Fix δ, η, γ. WTS: $HT_{[A/X]B}\ \underbrace{(\hat{\gamma}|M[A]|)}_{(\hat{\gamma}|M|)(\downarrow)}(\delta,\eta]$

By compositionality: STS:

$$HT_B((\hat{\gamma}|M|)(\downarrow))[\delta[X \mapsto \overbrace{A}^{\delta'}], \eta[X \mapsto \overbrace{HT_A(-)[\delta,\eta]}^{\eta'}]]$$

need to show this is a CR candidate.

By the IH,

$$HT_{\forall X.B}(\hat{\gamma}|M|)[\delta,\eta]$$

$\therefore HT_B((\hat{\gamma}|M|)(\downarrow))[\delta',\eta']$, and we're done ✓ □

Cor: $HT_{\Delta,\Phi}(M)[\phi,\phi]$ implies M term$_\beta$.

(To get more than just ans, use pair def'ns and t.c. = h.e. + term).

The critical point that makes this unjoyous in arithmetic is the set of sets quantifier. This termination proof can be used to show the consistency of 2nd order arith. The third-order aspect in the proof is when we said "for all type candidates"

Plan going forward:

wlo poly 1. Equality: formal / structural / semantic / behavioral

2. Can do parametricity / data abstr by binarising the foregoing arguments

3. Propositions-as-types ⟶ dep Types
4. Dep types, PCTT
5 HDTT

Data Abstraction / program verif

signature	reference impl.	candidate impl.
type t	type t = A_1	type t = A_2
val f : t→t	val f = M_1	val f = M_2
val c : t	val c = N_1	val c = N_2

$M_1 : t \to t \; M_2$ [t = R)
$N_1 : t \; N_2$ (t = R)

$\exists t.(t \to t) \times t$

$\triangleq \forall u.(\forall t.(t \to t) \to t \to u) \to u$

Client is polymorphic and the polymorphism ensures the implementation respects the relation relating the ref & cand implementations

⇒ Client can't tell the implementations apart

⇒ impl/repn independence

Back to simple types: ans →
(Could throw in 0, 1, ×, +, nat if you would)

1. Formal/structural TT
inductively defined i.e.,
$\Gamma \vdash M : A$ interpreted — a mapping over open terms.

Variables are indeterminates, calculation w "poly'ls"; open terms

This is a proof theory a means of accessing the truth.

→ has no notion of programs

→ Calculation done via $\Gamma \vdash M \equiv N : A$ (where $\Gamma \vdash M, N : A$)

$\equiv$ is the equality given by Gentzen's inversion principle: "elim cancels intro"

2. Semantic / Extensional TT.

$\Gamma \gg M \in A$, i.e., if $HT_p(\gamma)$, then $HT_A(\bar{\gamma}(M))$.

These two are related by FTLR.

This TT is a truth theory about programs

→ start w programs, types are specs.

→ Defn of $\Gamma \vdash M \equiv N : A$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash M \equiv M : A} \qquad \frac{\Gamma \vdash M \equiv N : A}{\Gamma \vdash N \equiv M : A} \qquad \frac{\Gamma \vdash M \equiv N : A \quad \Gamma \vdash N \equiv P : A}{\Gamma \vdash M \equiv P : A}$$ } equiv rules

$$(\xi)\ \frac{\Gamma, x : A \vdash M \equiv N : B}{\Gamma \vdash \lambda x{:}A.M \equiv \lambda x{:}A.N : A \to B}$$

$$\frac{\Gamma \vdash M \equiv M' : A \to B \quad \Gamma \vdash N \equiv N' : A}{\Gamma \vdash MN \equiv M'N' : B}$$

$$\frac{\Gamma, x : A \vdash M : B \quad \Gamma \vdash N : A}{\Gamma \vdash (\lambda x{:}A.M)(N) \equiv [N/x]M : B}$$ } universal

Notice: Very strict! And no hypothetical reasoning.

(Suppose you also had the rules for $\to$.)

Then you can define double & plus.

But

$\lambda x.\, nat.\, double(x)$

$\neq$

$\lambda x.\, nat.\, plus(x, x)$

} beware, not equality!

even though $double(\bar{m}) = plus(\bar{m}, \bar{m})$ for any m you want.

Similarly:

$\lambda x\ nat.\ plus(x, \bar{0}) \equiv \lambda x : nat.\ x$

but $\lambda x.\, nat.\, plus(\bar{0}, x) \not\equiv \lambda x.\, nat.\, x$

2018-02-15

Equality, heading to dependency.

Last time: In formal type theory, we have def'l equivalence

$$\Gamma \vdash M \equiv N : A$$

axiomatized as the least congruence containing β rules. ~~Never~~ Methodologically, intended to be decidable (never efficiently so, as Statman showed). An issue is, e.g., Shannon expansion.

In behavioural/semantic type theory, we have a full-throated notion of equality.

Motivation "Binarize logical relations"

FTLR: if $\Gamma \vdash M : A$, then $\Gamma \gg |M| \in A$, where $\Gamma \gg M \in A$ means $\forall \gamma$, $HT_\Gamma(\gamma)$ implies $HT_A(\hat{\gamma}(M))$.

Want at a min: if $\Gamma \vdash M \equiv N : A$, then $\Gamma \gg |M| \doteq |N| \in A$

"exact equality"

Also want a characterization of exact equality in terms of program behaviour.

Notation:

$M \in A$ means $HT_A(M)$

$\Gamma \gg M \in A$ means $\forall \gamma \in \Gamma$, $HT_\Gamma(\gamma)$ implies $HT_A(\hat{\gamma}(M))$.

$M \doteq N \in A$ means $EQ_A(M, N)$

idea: "binary HT"

Then redefine $M \in A$ means $M \doteq M \in A$.

$\doteq$ will be shown to be a PER

$\Gamma \gg M \doteq M' \in A$ means extensional / functional

maps (open terms) are functions.

Explicitly, if $\gamma \doteq \gamma' \in \Gamma$, then $\hat{\gamma}(M) \doteq \hat{\gamma}'(M') \in A$

e.g., $x:A \gg M \in B$ defines a function that respects exact equality in $x:A$:

if $P \doteq Q \in A$, then $[P/x]M \doteq [Q/x]M \in B$.

Let's go through all of the judgments again:

(critical induction)

$M \doteq M' \in \text{ans}$ iff either $M, M' \Downarrow \top$ or $M, M' \Downarrow \bot$ (being inductive)

$M \doteq M' \in \text{nat}$ iff either $M, M' \Downarrow 0$, or $M \Downarrow s(N)$, $M' \Downarrow s(N')$ and $N \doteq N' \in \text{nat}$

$M \doteq M' \in A_1 \to A_2$ iff $M \Downarrow \lambda x.M_2$, $M' \Downarrow \lambda x.M_2'$, and if $M_1 \doteq M_1' \in A_1$, then $[M_1/x]M_2 \doteq [M_1'/x]M_2' \in A_2$.

Fact 1) $M \doteq M' \in A$ implies $M' \doteq M \in A$.

2) $M \doteq M' \doteq M'' \in A$ implies $M \doteq M'' \in A$.

Pf: By induction on A. Use determinism of reduction in 2) $A_1 \to A_2$ (IH) and fact = is a PER

Aside parametricity = semantic equivalence for polymorphic types.

$M \doteq M' \in A$ iff Girard's Method / Reynold's Method

Funny: you can't directly show transitivity of parametricity. The reason is: when you use Girard's method, you're no longer working with equivalence rel'ns. You're working in heterogeneous rel'ns, they're then simulation rel'ns. You want to show the equivalence of candidates & reference impls to clients. If you understand this, you understand data abstraction, and so long else is data abstraction.

Fact (head expansion):
if $M \doteq M' \in A$ and $N \longmapsto^* M$ and $N' \longmapsto^* M'$, then $N \doteq N' \in A$.

Thm: Definitional equivalence suffices for exact equality.

Pf: By induction on the derivation.

eg) $\dfrac{\Gamma \vdash M : A}{\Gamma \vdash M \equiv M : A}$ FTLR+ if $\Gamma \gg M \in A$ then $\Gamma \gg M \doteq M \in A$.

eg) $\dfrac{\Gamma \vdash M \equiv M' : A}{\Gamma \vdash M' \equiv M : A}$ $\Gamma \gg M \doteq M' \in A$ $\therefore \Gamma \gg M' \doteq M \in A$ (? We haven't proven this!)

Lemma If $\Gamma \gg M \doteq M' \in A$, then $\Gamma \gg M' \doteq M \in A$.

Pf: Supse $\gamma' \doteq \gamma \in \Gamma$. Because γ, γ' are closed, know $\gamma \doteq \gamma' \in \Gamma$. WTS: $\hat{\gamma}'(M') \doteq \hat{\gamma}(M) \in A$
By assumption, $\hat{\gamma} M \doteq \hat{\gamma}' M' \in A$, so by sym, ☐

eg) $\dfrac{\Gamma, x:A_1 \vdash M_2 : A_2 \quad \Gamma \vdash M_1 : A_1}{\Gamma \vdash (\lambda x:A_1.M_2)M_1 \equiv [M_1/x]M_2 : A_2}$

Suppose $\gamma \doteq \gamma' \in \Gamma$. TS: $(\lambda x:A.\hat{\gamma}(M_2))(\hat{\gamma}(M_1)) \doteq [\hat{\gamma}'(M_1)/x]\hat{\gamma}'(M_2) \in A_2$.

① $FTLR_2$: $\hat{\gamma}(M_1) \doteq \hat{\gamma}'(M_1) \in A_1$
$FTLR_1$: if $N_1 \doteq N_1' \in A_1$, then $[N_1/x]\hat{\gamma}(M_2) \doteq [N_1'/x]\hat{\gamma}'(M_2) \in A_2$.
Instantiate N_1 and N_1' w ① and use head-exp.

eg) $\dfrac{\Gamma \vdash M \equiv M' : A_1 \to A_2 \quad \Gamma \vdash M_1 \equiv M_1' : A_1}{\Gamma \vdash M(M_1) \equiv M'(M_1') : A_2}$

Suppose $\gamma \doteq \gamma' \in \Gamma$. TS: $\hat{\gamma}(M)(\hat{\gamma}(M_1)) \doteq \hat{\gamma}'(M')(\hat{\gamma}'(M_1')) \in A_2$.
Use: **Lemma** If $M \doteq M' \in A_1 \to A_2$ and $M_1 \doteq M_1' \in A_1$, then $M(M_1) \doteq M'(M_1') \in A_2$.
Pf: Use head exp. ☐

Fact Assume $x:nat \gg M, M' \in A$. WTS when $x:nat \gg M \doteq M' \in A$. Claim.
STS: 1) $[0/x]M \doteq [0/x]M' \in A$
2) if forall $N, N' \in nat$, if $N \doteq N' \in nat$ and $[N/x]M \doteq [N'/x]M' \in A$, then $[s(N)/x]M \doteq [s(N')/x]M' \in A$.
RwH is pretty sure the above is right, but check it.

You might need to change 1) to

1) if $N \Downarrow 0$ and $N' \Downarrow 0$, then $[N/x]M \doteq [N'/x]M' \in A$

and update 2) in a similar manner.

Remark: We aren't using mathematical induction, but rather the inductive structure of the type nat.

Remark: 1) and 2) look a lot like the functionality statement, so we can use pseudo-notation and say:

2')

$$[y/x]M \doteq [y/x]M' \in A[=_{y:\mathrm{nat}}] \; [s(y)/x]M \doteq [s(y)/x]M' \in A.$$

(functionally/generically in y)

We can use this as a way state. What does this actually mean? That a certain fake hypothetical judgment holds:

$$y:\mathrm{nat},\ "[y/x]M \doteq [y/x]M' \in A" \gg [s(y)/x]M \doteq [s(y)/x]M' \in A.$$

You can get this if you treat equations as types and have realizers for them:

$$y:\mathrm{nat},\ z:\mathrm{Eq}_A([y/x]M, [y/x]M') \gg \ldots \in \mathrm{Eq}_A([s(y)/x]M, \ldots).$$

Remark: Eq_A is a type in terms in it. This is where dependent type come in.

We'll make these hand-wavy intuitions precise next week.

2018-02-20

Last time

1. Equational reasoning in formal and semantic type theory
   eg) $M \doteq N \in \mathbb{N}at \supset P(M,N)$

   STS) a) if $M, N \Downarrow 0$, then $P(M,N)$
   b) if $M, N \Downarrow s(M'), s(N')$ and $P(M',N')$, then $P(M,N)$
2. broached topic of props-as-types principle, heading towards dep type.

Modes of reasoning are very similar to the ~~history~~ modes of reasoning used to define → type construction

proofs-as-types aka proofs-as-progms

unification of logic and types.

Two key players:

1. Brouwer – developed a theory of truth based on computation } Church Martin-Löf
2. Hilbert – developed a theory of formal proof } Gentzen

Jan Heijenoort From Frege to Gödel

semantics vs.	syntactics
truth	proof
derived from the incohate understanding of algorithms	axiomatics consistent?

What does it mean for $M \in A \supset B$ to be true? ~~State~~ Brouwer: ~~the~~ M is an effective procedure that takes evidence for A to evidence for B.

Hilbert gave an inductive definition
M : A. The idea is that M is a
formal derivation ~~that~~ of A true.

M : A is by definition/design meaningless.
M ∈ A is by defn meaningful.

RWH: formal proof is at most a
sufficient condition for truth.

Formal proof (in logic)

1. Hilbert-type systems (combinators)
2. Gentzen-type system (λ-terms)

Hilbert system - emphasis on unconditional truth.

$A \supset A$ true
$A \supset (B \supset A)$ true
$(A \supset B \supset C) \supset (A \supset B) \supset (A \supset C)$

+ MP.

Gentzen-type system
Based on entailment
and hypothetical reasoning
Uses natural deduction

block structure

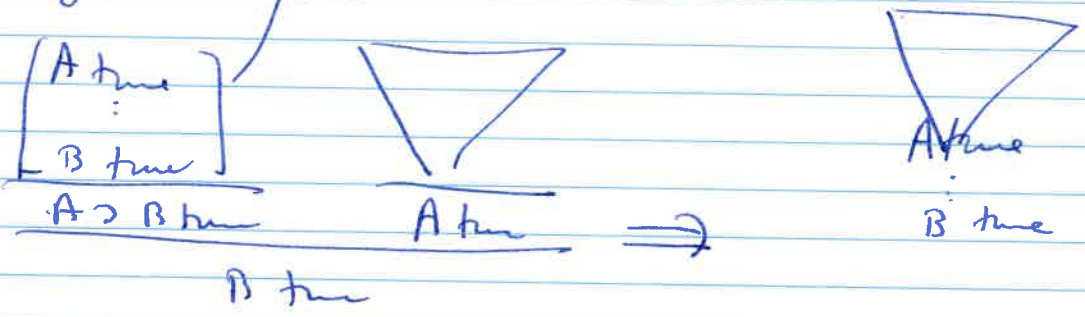
$$\frac{\begin{array}{c}[\text{type } A \text{ true}] \\ \vdots \\ B \text{ true}\end{array}}{A \supset B \text{ true}}$$

entailment

$$\frac{A \text{ true} \vdash B \text{ true}}{A \supset B \text{ true}}$$

Proof simplification

Gentzen: You have a rule



When are two proofs the same?
When are two algos the same?

Proof simplification is bound a substitution

Church: λ-calculus
Based on substitution

Formal corresp btw λ-calc + pfs.

Curry Should be renamed to
Church-Gentzen Correspondence.

This is a meaningless corresp, and only of historical relevance.

Formal corresp:
1. Pre-existing logics corresp to typing systems for λ-terms
2. w/ Gentzen's inputs induced a notion of $\beta\eta$ equiv same equiv functions.

Today: both formal logic & f. type systems are given by defining

$$\Gamma \vdash M : A$$

Justified if entailment is formal.
"$\Gamma \vdash M = N : A$" is equivalent justification

Until Gentzen, there was no notion of equality, and so the corresp doesn't even line up properly! Plus, what prop does "nat" corresp to?

The "Curry-Howard Isomorphism" is a meaningless corresp between a meaningless thing & something of interest. Only of historical note.

Brouwerian conception

Theory of truth, not formal proof
based on proof
which is algorithmic,
a human dialog.

p.M-L Judgments $M \in A$ M is a proof of A
evidence for constructive understanding of truth of A.

Semantic equality $M \doteq N \in A$.

Meaning of $M \in A$ (and why we've cared so much abt termination):
1) M evaluates to a value (can. form)
2) that value obeys the specification given by A.

the notion of truth depends on termination!

The meaning of $M \doteq N \in A$ is also based on running code / computation.

Semantic Corresp (Props-as-types)

$\top$	1	Nat
$\bot$	0	$\bar{A}$ (partiality)
$A \supset B$	$A \to B$	$\vdots$
$A \wedge B$	$A \times B$	
$A \vee B$	$A + B$	
$\neg A = A \supset \bot$	$A \to 0$ (A cont)	

There are many types that have nothing to do in logic and TT comes first.

E.g. $\bar{A}$ is partial fun and hence its inhabs need not terminate. ∴ No corresp to truth.

~~Unfamous intro.~~

$A \vee \neg A$ (for any A)
$A + (A \to 0)$

A is either true or false
provable refutable
and you know which it is.

2 things that were mixed ~~but~~ in each

1. Weaker disjunction: $\neg\neg(A \vee \neg A)$
or $\neg(\neg A \times \neg\neg A)$.

You can recover everything from Classical L, it just depends which version of $\vee$ you actually meant.

2. You can postulate an oracle

$$\mathcal{O}_0 \in A_0 + (A_0 \to \bot)$$

You can't run these proofs, but you can't run the classical proofs either.

Types / IL ~~Epis~~ is no more restrictive than CL.

Point: Too much is made of the formal corresp. The real one is the semantic corresp.

2018-02-22

Formal Dependent Type Theory

Terminology: often called "ITT" (Intentional Type Theory)

This is a formal type theory — inductively defined.

Its key idea is using type-indexed families of types. From a props-as-types perspective, these capture predicates/relations.

In particular: $I_A(M,N)$, where $M,N \in A$
$\mathrm{Vec}(M)$ where $M : \mathrm{nat}$

Today: "core" ITT with $\Sigma, \Pi, \mathrm{Id}_A$.

Syntactic judgments look like those we had before.

$\Gamma \vdash M : A$ $\qquad$ $\Gamma \vdash A \ \mathrm{type}$ $\qquad$ $\Gamma \ \mathrm{ctxt}$

$\Gamma \vdash M \equiv M' : A$ $\qquad$ $\Gamma \vdash A \equiv A'$ $\qquad$ $\Gamma \equiv \Gamma'$

Warning: the details are delicate and will be ignored. See Martin Hofmann's survey paper for these.

Structural Properties (axiomatically)

$$\frac{}{\Gamma, a:A, \Gamma' \vdash a : A}\ (R)$$

The key point is that "a" can appear in the type in ~~Γ~~ Γ'. The contexts are ordered.

Remark that A is a type ~~in~~ in the ctxt Γ, and it remains a type when we weaken with Γ'.

$$\frac{\Gamma \vdash A \ \mathrm{type} \qquad \Gamma \vdash B \ \mathrm{type}}{\Gamma, a:A \ \mathrm{type} \vdash B \ \mathrm{type}}\ (W)$$

"B type" could be replaced by any atomic judgment $\mathcal{J}$

"order doesn't matter (P) unless it does" $\equiv$ is R, S, T and compat w/ constructors

$$\frac{\Gamma, a:A, b:A, \Gamma' \vdash J}{\Gamma, z:A, [z,z/a,b]\Gamma' \vdash [z,z/a,b]J} \quad (C)$$

$$\frac{\Gamma \vdash M:A \qquad \Gamma, a:A, \Gamma' \vdash J}{\Gamma[M/a]\Gamma' \vdash [M/a]J} \quad (S)$$

(Trans / Cut)

Variables are treated structurally, instead of substructurally.

(Aside: In Church's original λ-calculus, there was no weakening, so you couldn't write $\lambda x.\lambda y.\, x$)

(At Pitt in the 1960s, Nuel Belnap and his students developed relevance logic an early investigation into substructural logic, where you could use a variable as many times as you wanted, as long as you used it at least once.)

Invariance

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A \equiv A'}{\Gamma \vdash M : A'} \quad M \equiv M'$$

What should $A \equiv A'$ be? It could be empty, but choosing a "good" $\equiv$ drives everything. It makes no difference in the simply-typed setting, but in the dependently typed setting it really matters and is induced by term equivalence.

Booleans

Surprise! Hard to capture.

$$\frac{\Gamma \text{ ctxt}}{\Gamma \vdash \text{Bool type}} \qquad \frac{}{\Gamma \vdash tt : \text{Bool}} \qquad \frac{}{\Gamma \vdash ff : \text{Bool}}$$

Want to write:

$$\frac{\Gamma \vdash M : \text{Bool} \quad \Gamma \vdash P : A \quad \Gamma \vdash Q : A}{\Gamma \vdash \text{if}(M; P; Q) : A}$$

$$\frac{}{\begin{cases} \Gamma \vdash \text{if}(tt; P; Q) = P : A \\ \Gamma \vdash \text{if}(ff; P; Q) = Q : A \end{cases}}$$

P.S. Do we want the Shannon expansion? (No)

The reason this is insufficient is that we
1) usually want to be able to read the type of a term. But ↳ B/C of decidability

$$\text{if}(M; P; Q)$$

Doesn't have a clear type: You could read

$$\Gamma \vdash P : A \quad \text{and} \quad \Gamma \vdash Q : A'$$

and then what do you do?
So we add a type annotation

$$\text{if}_A(M; P; Q).$$

2) Dependency. Normally want an open world and not have to update one constr from others.

Suppose, indeed, that you have

$\exists a : Bool$

$$\frac{\Gamma \vdash a : Bool \qquad \Gamma \vdash P : A \qquad \Gamma \vdash Q : A}{\Gamma \vdash if_A(a; P; Q) : A}$$

Here P and A and Q could depend on A. So you want to generalize the type of the conditional and propagate upstairs.

$$\frac{\Gamma \vdash M : Bool \qquad \Gamma, a : Bool \vdash A \text{ type} \qquad \Gamma \vdash P : [tt/a]A \qquad \Gamma \vdash Q : [ff/a]A}{\Gamma \vdash if_{a.A}(M; P; Q) : [M/a]A}$$

What are meaningful examples of $a.A$? Well, you might want to write something of the form

$$if_{a.?}(M; 17; \text{"17"}) : \underbrace{If(M, Nat, Str)}_{???}$$

What is this? You need a distinction between terms and types, so you instead use a "Large Elim"

$$\_\_\_\_\_ : If(M, Nat, Str).$$

This problem is not restricted to Bools, but rather all positive types. Types is a mapping our property.

3) "Large elims": mapping into type.

4) The issue w the Shannon exp is: it's the universal property for the Bools

But you can't get this in the formal setting (57)

Π, Σ

There's an issue as to whether you treat Σ as pos or neg. We'll treat Σ negatively.

Before: $\dfrac{\Gamma \vdash A \text{ type} \quad \Gamma \vdash B \text{ type}}{\Gamma \vdash A \underset{\to}{\times} B \text{ type}}$.

Now:

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma, a:A \vdash B \text{ type}}{\begin{array}{ll} \Gamma \vdash a:A \to B \text{ type} & (\Pi x:A.B) \\ a:A \times B \text{ type} & (\Sigma x:A.B) \end{array}}$$

NuPRL's syntax.

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma, a:A \vdash B \text{ type} \qquad \Gamma, a:A \vdash M:B}{\Gamma \vdash \lambda a:A.M : a:A \to B}$$

$$\frac{\Gamma \vdash M : a:A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M(N) : [N/a]B}$$

$$\frac{\cdots \quad ---}{\Gamma \vdash (\lambda a:A.M)N \equiv [N/a]M : [N/a]B}$$

Won't be able to write $\langle M, N \rangle$, what you'll really mean is something like

$$\underset{A, a.B}{\text{pair}}(M, N).$$

But lazy, so:

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma, a{:}A \vdash B \text{ type} \qquad \Gamma \vdash M : A \qquad \Gamma \vdash N : [M/a]B}{\Gamma \vdash \langle M, N \rangle : a{:}A \times B}$$

Negative formulation of Σ:

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma, a{:}A \vdash B \text{ type} \qquad \Gamma \vdash M : a{:}A \times B}{\left\{ \begin{array}{l} \Gamma \vdash \mathrm{fst}(M) : A \\ \Gamma \vdash \mathrm{snd}(M) : [\mathrm{fst}(M)/a]B \end{array} \right\}}.$$

$$\left\{ \begin{array}{l} \Gamma \vdash \mathrm{fst}(\mathrm{pair}_{A,a.B}(M, N)) \equiv M : A \\ \Gamma \vdash \mathrm{snd}(\text{——————}) \equiv N : [M/a]B \end{array} \right.$$

Identity Types

Only now do we know what's really going on, though this has been a vexed question for ages.

Idea: internalize "equality".

1. Defl equivalence is defined for typed terms, but is not type dependent.
2. RWH forgot. (DK)
3. "Judgmental concepts precede type concepts".

Just like you have the judgmental notion of map

$$\Gamma, a : A \vdash M : B$$

which you internalize as

$$\Gamma \vdash \lambda a : A.\, M : a : A \to B,$$

you want to do the same for equality.

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash M, N : A}{\Gamma \vdash Id_A(M, N) \text{ type}}$$

What is $Id_A(M,N)$? It's the least reflexive relation. This isn't what you think it is though and has been the cause of much consternation.

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash M : A}{\Gamma \vdash \underline{refl_A(M)} : \underbrace{Id_A(M,M)}_{\text{~~family of types~~}}} \quad (I)$$

$Id_A(M,N)$ is a family of types over $A \times A$. For particular choices of M, N, it might not be populated.

This type is a subsingleton. at most 1 elem

We use this terminology instead of saying "0 or 1 inhabitants" because we don't necessarily know which

The leastness is given by a mapping out property. Cannot McBride's / terminology for the induction

$\Gamma, a:A, b:A, c:Id_A(a,b) \vdash C$ type "the motive"
$\Gamma \vdash P : Id_A(M,N)$
$\Gamma, a:A \vdash Q : [a,a,refl_A(a)/a,b,c]C$

$$\Gamma \vdash J_{a,b,c.C}(a.Q)(P) : [M,N,P/a,b,c]C$$

. . . . .

$$J_{a,b,c.C}(a.Q)(refl_A(M)) \equiv [M/a]Q$$
$$: [M,M,refl_A(M)/a,b,c]C.$$

What rel'n is it?

$P : Id_A(M,N)$ // what?

Fact

1) Symmetric:

$$a : Id_A(M,N) \vdash \overset{sym_{A,M,N}(a)}{\cancel{sym(a)}} : Id_A(N,M)$$

2) Transitive:

$$a : Id_A(M,N), b : Id_A(N,P) \vdash \overset{a.b}{/} : Id_A(M,P)$$

$$trans_{A,M,N,P}(a,b)$$

Exercise: Find sym & trans.

Trans: Use Yoneda.

Ex: Show that the following "equalities".

$\mathrm{Id}_{\mathrm{Id}_A(M,M)}(\mathrm{sym}(\mathrm{refl}(M)), \mathrm{refl}(M))$.

$\mathrm{Id}_{\mathrm{Id}_A(M,N)}(\mathrm{trans}(\mathrm{refl}(M), u), u)$

:

Groupoid laws.

2018-02-27

Notation is cumbersome, so we write $- =_A -$ for the type $\mathrm{Id}_A(-,-)$

$=_A$ is an equivalence relation.

1. Reflexivity

$\Gamma, a:A \vdash a =_A a$ true

witness: $\mathrm{refl}_A(a)$.

2. Symmetry:

$\Gamma, a:A, b:A, p: a =_A b$ true $\vdash$ $b =_A a$ true.

Witness:

$\mathrm{sym}(a)(b)(p) : b =_A a$

often abbreviated as p^{-1}.

$J[a, b, \_. b =_A a](a, \mathrm{refl}_A(a), p) :$

$: b =_A a \equiv C[a, b, p]$.

Note: $\mathrm{sym}(a)(a)(\mathrm{refl}_A(a)) \equiv \mathrm{refl}_A(a)$ is a consequence of the exact term used to define symmetry.

3. Transitivity:

$a, b, c : A,\ p : a =_A b,\ q : b =_A c \vdash a =_A c$ true

Witness: $trans(a)(b)(c)(p)(q)$:

Notation: ~~$(q \circ p)$~~ $p \cdot q$

Auxiliary:

$a, b : A,\ p : a =_A b \vdash (C : A \to (b =_A c \to a =_A c))$ true

(like power version of Yoneda)

$J[a, b, \_, C : A \to (b =_A c \to a =_A c)](a, \lambda c : A.\ id^{?}_{a =_A c};\ p)$

$\lambda x.x$

$: C[a, b, p] \equiv c : A \to (b =_A c \to a =_A c)$

$c : A \to (a =_A c \to a =_A c)$

Then apply this to c and q.

Note: $trans(b)(b)(c)(refl_A(b))(q) \equiv q$.

$trans(a)(b)(b)(p)(refl_A(b)) \not\equiv p$

Summary

$refl_A(M)$ aka $id(M)$

p^{-1} symmetry

$p \cdot q$ trans

$id(M)^{-1} \equiv id(M)$

$id(M) \cdot q \equiv q$

Tantalizing thought: this looks like group structure.

1. Respect for identity aka transport, aka ~~Leib~~ Leibniz's Principle of indiscernibility of identicals

transport

$$\frac{\Gamma \vdash p : M =_A N \qquad \Gamma, a:A \vdash B \ \text{type}_{\text{prop}}}{\Gamma \vdash tr[a.B](p) : B[M] \to B[N]}$$

→ Definable from J such that

$$tr[a.B](refl_A(M))(R) \equiv R$$

(You need to think about these things: though it behaves like identity, whether or not it actually equals the identity depends on how you wrote the code)

$$tr[a.B](p) \triangleq$$

$$J[a, b, -.B[a] \to B[b]](a.\underbrace{id_{B[a]}}_{B[a] \to B[a]}; p]$$
$$: C[M, N, p] \equiv B[M] \to B[N].$$

Because of the way you wrote the code, you get

$$tr[a.B](refl_A(M)) \equiv refl_{B[M]},$$

but this is purely accidental

∴ "Id is a notion of equality".

Or is it?

Consider the following:

(✗) $b : Bool \vdash if_{-.Bool}(b; tt; ff) =_{Bool} b$ — type

Need a term of the stated "equality" type. (Can check this)

But it is not the case that

$$\vdash \lambda b:Bool.\ if_{-.Bool}(b; tt; ff) =_{Bool \to Bool} \lambda b:Bool.\ b \quad true$$

Why? M-L's Mom (approx 1977):

$M =_A N$ true iff $M \equiv N : A$.

And you can show that the two functions are not definitionally equal.

This is weird because then $A \to B$ is not a function type (like you learned in school).

So if you try to mechanize mathematics using Id as your notion of equality then you are screwed.

(Ex) Proof is by induction on b.

$$\mathrm{if}_{(a.\ \mathrm{if}_{-.\mathrm{Bool}}(a; \mathrm{tt}, \mathrm{ff}) =_{\mathrm{Bool}} a)}(b; \mathrm{refl}_{\mathrm{Bool}}(\mathrm{tt}); \mathrm{refl}_{\mathrm{Bool}}(\mathrm{ff})) : \mathrm{if}_{-.\mathrm{Bool}}(b; \mathrm{tt}; \mathrm{ff}) =_{\mathrm{Bool}} b.$$

□

Terminology: You do not have function extensionality

intensional ≜ not (extensional).

What to do?

0) Be happy: Id is internalized definitional equality.

1) But: I want Bool → Bool to be a type of functions (Boor you! The secret to happiness is not to want too much).

a) Go straight to setoid hell.

setoid = type + equiv reln
ways = respect equivalence.

Then take

$$(Bool \to Bool)/E$$

where

$E(F,G)$ iff $x : Bool \vdash F(x) =_{Bool} G(x)$.

Laborious.

b) Cheat: ETT.

Postulate equality reflection

$$\frac{\Gamma \vdash P : Id_A(M,N)}{\Gamma \vdash M \equiv N : A}\,(ER)$$

and uniqueness of identity proofs.

$$\frac{\Gamma \vdash P, Q : Id_A(M,N)}{\Gamma \vdash P \equiv Q : Id_A(M,N)}\,(UIP)$$

Perfectly valid on closed terms'

Except type checking ≠ proof checking, undecidable now

which blows the entire dogma of ITT out of the water.

c) ruse: add new elements of $Id_A(-,-)$.

i) $FUNEXT_{A,B} : \Pi F, G : A \to B.\ (\Pi a : A.\ F(a) =_B G(a)) \to Id_{A\to B}(F,G)$.

Adding this constant/axiom breaks M-L's Mm.

Observation: What else is wrong? What's the comput. content? What is

(OTT) $J[-](-; FUNEXT_{A,B}(F,G,P)) \stackrel{?}{\equiv}$???

↳ ad-hoc answer to this.

(HOTT)

ii) Univalence Axiom (Voevodsky) (later)

"Bury in" some new term UA

$$Equiv(A,B,E) \longrightarrow Id_{\mathcal{U}}(A,B)$$

"isomorphism" ↳ universe

You still have the exact same problem.

2017-03-01

What would be a comput interp of ITT ~~VL~~?

Informally, want to run (?) closed terms (no free vars).

1. Erasure — types are irrelevant at run time
   - ↳ Extraction — recovering a "raw program" from a derivation

2. Deterministic operational semantics, head reduction

Notice: Transport is a no-op at run-time
→ All the identity hacking amounts to nothing.

Key issue with ~~OTT~~ HoTT is that this idea of computational content does not work.

Let's examine the computational content of ITT.

"Computational Meaning Explanation" — FTLR.

Recall

1. We gave a semantics in terms of computation for $\lambda\to$, which did not have dep types.
   - If $\Gamma \vdash M : A$, then $\Gamma \gg |M| \in A$
   - Tait's, Girard's, and Reynold's methods.

NB: Start with programs. Extraction is fundamental.

How do we generalize this to dependent types (ITT)?

Crucial idea: Types are programs. They run and yield ~~programs~~ specifications.

Eg: if $(M, 17, \text{"17"}) \in$ if (M, Nat, Str).

You need to give up the phase distinction.

Consequently, the meaning explanation must generalize substantially for running types as ~~with~~ programs.

We must generalize the meanings as follows.

$\Gamma \vdash A$ type implies $\Gamma \gg A$ type

Where $\Gamma \gg A$ type means at least that A is well-behaved as a program. By abuse of notation, "$HT_{Type}(A)$". Also

$\Gamma \vdash A \equiv B$ type implies $\Gamma \gg A \doteq B$ type.

"$EQ_{Type}(A, B)$".

Computational Dep Type Theory

Given an op sem for closed erased terms

$A, M \Downarrow V$ V val } deterministic

$A, M \mapsto^* V$

Define the ff semantic judgments:

A type	"I know that A is a type"
$A \doteq A'$	"I know that A and A' are equal type"
$M \in A$	"Given that I know A type, I know that M exhibits the behaviour specified by A/ satisfies A"
$M \doteq M' \in A$	"... M and M' equisatisfy A"

(All erased terms)

A type means $A \Downarrow V$ and V is/names a specification (of behaviours)

The litterature says "V is a canonical type", where "canonical type" should be read as a noun "kanonical-type", rather than as ~~adj + noun~~. adj + noun, i.e., a special ~~kn~~ sort of "type".

eg) Bool is a specification (see below)
Bool → Bool
if(M, Nat, String) evaluates to a spec.

$A \doteq A'$ means $A \Downarrow V$, $A' \Downarrow V'$, V and V' are equivalent/equal specs.

$M \in A$ where $A \Downarrow V$ spec
$M \Downarrow W$ and W satisfies V
"W is a canonical element of V"

eg) tt sats Bool
ff sats Bool
(nothing else)
} Bool is inductive.

$M \doteq M' \in A$ where $A \Downarrow V$ spec
$M \Downarrow W$, $M' \Downarrow W'$, W and W' equisatisfy V.

(Subsumes def'l equivalence)

Eg: $id \doteq abs \in Nat \to Nat$
$id \neq abs \in \mathbb{Z} \to \mathbb{Z}$.

All of this can be consolidated as

- $A \doteq A'$ equal specs
- $M \doteq M' \in A$ equally satisfy specs.

Will be sym + trans, and therefore refl on their field (PERs)

- V and V' are equal specs.
- V and V' eq. sat W

Eg: $\langle M, N \rangle \in A \times B$ (values)
iff $M \in A$ and $N \in B$ (non-values)

There are categorical judgments (stated w/o any conditions), not to be confused w categorial judgments.

We need hypothetical judgments
(in M-L: hypothetico-general)

→ Semantics of variables!
range over elements of their type!
(values?)
(Choice of elements vs values in CBN vs CBV)

1) $\bullet \gg J$ means J.
2) Given that A type,
A) $a : A \gg B$ type means
(first, and insufficient)
if $M \in A$ then $[M/a]B$ type
Want/need more, because types are defined by PERs, and you need to ensure this is respected. $M \in A$ does not pick out a canonical repr for the equiv class, it's just a piece of code.
So you need:

if $M \doteq M' \in A$, then $[M/a]B \doteq [M'/a]B$ type

And what you really mean, seeing that you're working in PERs, is that

~~if $M \doteq M' \in A$ then $[M/a]B \doteq [M'/a]B$ type~~ $a : A \gg B \doteq B'$ type means

if $M \doteq M' \in A$ then $[M/a]B \doteq [M'/a]B'$ type.

B) $a : A \gg N \doteq N' \in B$
(given that $a : A \gg B \doteq B$)

if $M \doteq M' \in A$ then
$[M/a]N \doteq [M'/a]N' \in [M/a]B$ $(\doteq [M/a]B)$

"FUNCTIONALITY" maps/open terms are functional (i.e. resp equality) in their free vars.

This generalizes in an inductive manner

$a_1 : A_1,\ a_2 : A_2(a_1),\ \ldots,\ a_n : A_n(a_1, \ldots, a_{n-1})$ ctx.

$a_1 : A_1, \ldots, a_n : A_n \gg A \doteq A'$
$M \doteq M' \in A$

"iterated functionality" (CMCP)

→ Check that the formal structural properties are validated as hypothetical

$$\frac{\Gamma, a : A, \Gamma' \gg N \in B \qquad \Gamma \gg M \in A}{\Gamma[M/a]\Gamma' \gg [M/a]N \in [M/a]B}$$

You'll discover that all of the properties of hypothetical judgements are true: we have more than just entailment, you're given a semantics of variables.

eg) Bool as a type

1. Bool and Bool are equal types.
2. tt and tt equally satisfy Bool
   ff and ff
   nothing else.

V and V' eq. for Bool if either

$V = V' = tt$
or $V = V' = ff$. (syntactically equal)

Fact This is not a definition!

It is a fact:

If $b : \text{Bool} \gg A \doteq A$ (motive)
$M \doteq M' \in \text{Bool}$
$N \doteq N' \in [tt/b]A$
$P \doteq P' \in [ff/b]A$,

then $\text{if}(M; N; P) \doteq \text{if}(M', N', P') \in [M/b]A$.

We will see a proof next time.

(7 4)

2018-03-06

Recall
hypothetical (general) judgments define what is a mapping — functionality.

$$x_1 : A_1, \ldots, x_n : A_n \vdash A \doteq A'$$
$$M \doteq M' \in A$$

$n=0$: categorical
$n+1$: consider

$$x_1 : A_1, \ldots, x_n : A_n \vdash A_{n+1} \text{ type}$$

define

$$x_1 : A_1, \ldots, x_{n+1} : A_{n+1} \vdash \begin{cases} A \doteq A' \\ M \doteq M' \in A \end{cases}$$

Consider all pairs of instances
if $M_1 \doteq M_1' \in A_1$
$M_2 \doteq M_2' \in [M_1/x_1]A_2$,
...
then $[M_i/x_i]_{i=1}^{n+1} A \doteq [M_i'/x_i]_{i=1}^{n+1} A'$

$$\frac{}{M \doteq} \quad \frac{}{M' \in [M_i/x_i]_{i=1}^{n+1} A}.$$

Check for categorical, and then hyp judgments
1. symmetry
2. transitivity
} (Check)

Use the "PER trick" if $M \doteq M' \in A$ then $M' \doteq M \in A$
then $M \doteq M \in A$.

Check structural properties of hypothetical corr to logical entailment

1) $\Gamma, x:A, \Gamma' \vdash x \in A$ (reflexivity)
2) if $\Gamma \vdash J$ and $\Gamma \vdash A$ type, then $\Gamma, x:A \vdash J$ (weakening)
3) if $\Gamma, x:A, \Gamma' \vdash J$ and $\Gamma \vdash M \doteq M' \in A$, then "$\Gamma [M/x]\Gamma \vdash [M/x]J$".

Once the computational framework is setup, we can see how logic is embedded.
1. Inhabitation
Γ, A true, $\Gamma' \vdash A$ true
etc
2. plus ~~compose~~ introduce / impose concept

of (exact) equality of proofs.
↳ Dependent on the prop!

This is the semantic props-as-types principle.

Next

Populate the theory w/ types.

1. $Bool \doteq Bool$.

Def'n: least type containing tt, ff.
→ properties of conditionals are derived from this definition.

2. $\Pi x{:}A.B$, aka $x{:}A \to B$
$\Sigma x{:}A.B$, aka $x{:}A \times B$.
Key. the elim forms are theorems/facts.

→ function equality is extensional (wrt $\doteq$)

$\lambda(x.M) \doteq \lambda(x.M') \in x{:}A \to B$ } equality of behaviour
iff $x{:}A \vdash M \doteq M' \in B$
closed values.
Respect for equality

Equality types

Internalize judgmental equality

$$F\left[\begin{array}{l} Eq_A(M,N) \doteq Eq_{A'}(M',N') \\ \text{iff} \\ \quad A \doteq A',\ M \doteq M' \in A \\ \quad\quad N \doteq N' \in A \end{array}\right.$$

$$\text{Intro}^{pp}_{\text{defn}}\ [\ * \doteq * \in Eq_A(M,N) \text{ iff } M \doteq N \in A$$

(73)

Could write "refl" instead of "$*$", and $|refl_A(M)| = refl$.

Note: $refl \in Eq_{Nat \to Nat}(\lambda a.\, 2 \times a, \lambda a.\, a + a)$

WTS: That the computational interp is a valid interp of the formal theory:

1) If $\Gamma \vdash M : A$ then $|\Gamma| \gg |M| \in |A|$

2) If $\Gamma \vdash M \equiv M' : A$, then $|\Gamma| \gg |M| \doteq |M'| \in |A|$

3) If $\Gamma \vdash A$ type, then $|\Gamma| \gg |A| \doteq |A'|$

4) If $\Gamma \vdash A \equiv A'$, then $|\Gamma| \gg |A| \doteq |A'|$

proof theory — Truth theory

Crucial Part

$|Id_A(M,N)| \triangleq Eq_{|A|}(|M|, |N|)$

$|refl_A(M)| \triangleq refl$

$|J[a,b,e.C](a, Q, P)| \triangleq [|M|/a]|Q|$
where $P : Id_A(M,N)$

↳ literally does nothing.

$|\Gamma, a:A, b:A, c:Id_A(b,c)| \gg |B|$ type

$|\Gamma|, a:|A|, b:|A|, c:Eq_{|A|}(a,b)$

$|\Gamma| \gg |P| \in |Id_A(M,N)| = Eq_{|A|}(|M|, |N|)$

$|\Gamma| \gg |M| \doteq |N| \in |A|$

$|\Gamma|, a:|A| \gg |Q| \in |B[a,e,refl_A(a)]| = |B][a,a,refl]$

Then $|\Gamma| \gg [|M|/a]|Q| \in |B|[|M|, |M|, |P|]$
(def $|J\ldots|$) $\quad$ $|B[M,M,P]|$

Why have we defined anything at all?

Bool

$a : A \to B$ ← Need to know: (A type, $a : A \gg B$ type; "Prior to $a : A \to B$"; $[M/a]B$; ∞-many such instances)

$\alpha : A \times B$

$Eq_A(M, N)$

Principles of definition of these?

→ Predicativity: "structural" induction. You need to know what came prior

→ "local def'ns" such as Bool.

2018-03-07

What exactly justifies the construction of a type system?

Recall: With simple types it's easy.

1. Define types $A ::= b \mid A_1 \to A_2 \mid \cdots$
2. Defined predicates by induction on the structure of $HT_{A_1 \to A_2} = HT_{A_1} \to HT_{A_2}$

With dependency, it's stickier because types are tied up with elements.

eg) $Eq_A(M, N)$ type when A type, $M, N \in A$

and $P \in Eq_A(M, N)$ when $M \doteq N \in A$.

With inductive definitions, another concern arises.

Stratified $\in_0$ then $\in$.

Recall Bool values are the least PER over tt, ff s.t.

$tt \doteq tt \in_0 Bool$ } define values

$ff \doteq ff \in_0 Bool$ }

(Then) Bool computations are

$M \doteq N \in Bool$ iff $M \Downarrow V$, $N \Downarrow W$ and $V \doteq W \in_0 Bool$

Nat is inductively defined as follows

$zero \doteq zero \in_0 Nat$

$succ(M) \doteq succ(M') \in_0 Nat$ when $M \doteq M' \in Nat$

$M \doteq M' \in Nat$ iff $M \Downarrow V$, $M' \Downarrow V'$, $V \doteq V' \in_0 Nat$.

Not $\in_0$!

Today we'll go after a more careful construction by Allen '87.

1. This should give us clarity going forward
2. We can "calibrate the metatheory": where does the construction take place? Deal w the infinite regress of justifications for the semantic model. "ordinal analysis" ≡ calibrate using ordinals.

Let's make explicit the inductive constructions that define computational dependent type theory.

→ We have fixed points for monotone operators, by the Knaster-Tarski theorem:

Let $\mathcal{L}$ be a complete lattice:

1. a pre-order $\leq_{\mathcal{L}} \subseteq |\mathcal{L}| \times |\mathcal{L}|$
2. all subsets have meets (glb) and joins (lub):
   if $X \subseteq \mathcal{L}$, then $\bigwedge X \in \mathcal{L}$ and $\bigvee X \in \mathcal{L}$.

A plain lattice has binary meets and joins $x \wedge y$, $x \vee y$, and $\top$ and $\bot$.

Then Tarki's thm says:

An order preserving function (monotone) $f : \mathcal{L} \to \mathcal{L}$ ($\mathcal{L}$ – complete lattice) has a complete lattice of fixed points. In particular, f has a least fixed point given by $\bigwedge \{x \in \mathcal{L} \mid f(x) \leq x\}$.

Concretely, $\mathcal{L}$ is often a powerset $\wp(A)$ ordered by inclusion, where meets are $\cap$ and joins are $\cup$.

When $F(X) \subseteq X$, we say "X is closed under F", or "X is F-closed".

We'll stratify our construction to leverage Knaster-Tarski into two levels: a Candidate type system, and then the real type system, which will be a well-behaved candidate in a sense to be explained.

0. We have a programming language w/ $M \hookrightarrow M'$ and $M\,\mathrm{Val}$ as judgments.

1. Candidate type system

$\tau(A, B, \varphi)$ — A, B — closed values
$\varphi(V, W)$ — V, W — values. "value set'in"

e.g.) τ tells you when you have "equal types" along w/ their equiv terms.

$\tau(\mathrm{Bool}, \mathrm{Bool}, \beta)$
$\beta(tt, tt)$, $\beta(ff)$, nothing else.

2. A Real type system is a well-behaved candidate in the following sense:

a) τ is functional:
if ~~$\tau(A,B)$~~ $\tau(A, B, \varphi)$ and $\tau(A, B, \varphi')$, then $\varphi = \varphi'$.
In other words, you don't get to assign two different term rel'ns to the same type.

b) PER-valued: if $\tau(A, B, \varphi)$, then φ is a value-PER (sym & trans)

c) type equality is a PER:
- if $\tau(A, B, \varphi)$ then $\tau(B, A, \varphi)$
- if $\tau(A, B, \varphi)$ and $\tau(B, C, \varphi)$, then $\tau(A, C, \varphi)$.

Remark: a CTS forms a complete lattice under inclusion.

3. Define a monotone operator on candidate type systems:

$$\Phi(\tau) \triangleq \mathrm{BOOL}(\tau) \cup \mathrm{EQ}(\tau) \cup \Pi(\tau) \cup \Sigma(\tau)$$

(7D.

Show monotonicity.

$\therefore T_0 := \mu\Phi$ least fixed point

is the type system defined by Φ.
Show T_0 is a real type system.

4. Define truth relative to a real type system

$\tau \vDash A \doteq B$ iff $\tau^{\Downarrow}(A, B, -)$
$\tau \vDash M \doteq N \in A$ iff $\varphi(M, N)$ when $\tau^{\Downarrow}(A, A, \varphi)$

where the liftings are defined as follows:
- if φ is a value relation,
  then $\varphi^{\Downarrow}(M, N)$ iff $M \Downarrow V$, $N \Downarrow W$, $\varphi(V, W)$
- if τ is a cand. type syst.
  then $\tau^{\Downarrow}(A, B, \varphi)$ iff $A \Downarrow V$, $B \Downarrow W$, $\tau(V, W, \varphi)$

5. Show that typical rules are true wrt T_0, the initial type system

eg) If $T_0 \vDash M \in A \to B$
And $T_0 \vDash N \in A$
Then $T_0 \vDash M(N) \in B$
⋮

Then $\to, \times, (\Pi, \Sigma)$ have their full universal properties.

Remark: You must accept Knaster-Tarski if you want to accept this construction. There are different constructions you could use that don't need K-T, but at the end of the day, you need to accept some form of inductive definition.

6. Make $\Phi(\tau)$ explicit:

- BOOL(−)(Bool, Bool, β) as before
- ~~EQ(τ)(A, B,~~

- $EQ(\tau)(A_0, A_0', \varepsilon)$ iff (defn)
  1. $A_0 = Eq_A(M,N)$ val
  2. $A_0' = Eq_{A'}(M', N')$ val
  3. $\exists \varphi$ s.t. $\tau^{\Downarrow}(A, A', \varphi)$
  4. $\varphi^{\Downarrow}(M, M')$ and $\varphi^{\Downarrow}(N, N')$
  5. $\varepsilon(\text{refl}, \text{refl})$ iff $\varphi^{\Downarrow}(M,N)$ (iff $\varphi^{\Downarrow}(M', N')$).

- $PI(\tau)(A_0, A_0', \rho)$ iff (def'n)
  1. $A_0 = x:A \to B_x$, $A_0' = x:A' \to B_x'$
  2. $\tau^{\Downarrow}(A, A', \alpha)$ $(\exists \alpha)$
  3. ~~if $\alpha^{\Downarrow}(M, M')$ then~~ $\exists \Psi : (Tm \times Tm \to ValRel)$ s.t. (Might need to require Ψ respect computation.)
     for all M, M', if $\alpha^{\Downarrow}(M, M')$, then $\tau^{\Downarrow}([M/x]B, [M'/x]B', \Psi(M, M'))$
  4. $\rho(M, M')$ iff (def'n)
     a) $M = \lambda a.N$, $M' = \lambda a.N'$
     b) if $\alpha^{\Downarrow}(P, P')$, then $\Psi(P,P')^{\Downarrow}([P/a]N, [P'/a]N')$.

- $SIGMA(\tau)$ is analogous.

Check: These are all monotone.

The gist / core idea is that "A and all $[M/x]B$ for $M \in A$ are prior to $x:A \xrightarrow{X} B$"

The $[M/x]B$ are not substructures of $x:A \to B$ and $x:A \times B$. But they are prior and are never lost when you extend the type system by applying Π.

If you try to formalise this, you need to be able to ~~load~~ handle inductively-defined families, to be able to grasp the above distinction.

Contrast

1. cf. $\forall X.A$. You cannot consider all $[B/X]A$ prior to $\forall X.A$! This is because B might be $\forall X.A$!

You can't add

~~POLY / ALL $(\tau)(A, B, \alpha)$~~

ALL/POLY(κ)(∀X.A, ∀X.A',) ... τ([B/X]A, [B/X]A')

2. Cannot "expand" "revise" the definition of any prior type by any monotone constr. Consider the (bad) X:

X(τ)(Nat, Nat, η')

where τ(Nat, Nat, η) and η' ⊉ η.

(~~and maybe~~ η ⊄ η' ?)

Because previous levels ~~depend~~ may have depended on the defn of Nat, and by changing Nat you may invalidate them.

Pare spring break 2018-03-20

1.5 weeks ago, we saw pre-(type systems) candidate type systems
Recall we never define an F(τ) where F(τ) is defined ~~if~~ by a condition
"if F(τ) (negative) then F(τ)". → you loose monot.
Only ever of the form "if τ then F(τ)" (positive).

We defined

$$\tau_0 = \mathrm{lfp}\ \Phi$$

(renamed Φ to J today).

τ_0 is a pre-(type system) by construction.

1) functionality.

TS: if $\tau_0(A, B, \varphi)$ then $\varphi = \varphi'$ (and) then if $\tau_0(A, B, \varphi')$ ($\varphi(M,N)$ iff $\varphi'(M,N)$)

(Note, A, B, M, N are values b/c we haven't lifted φ, τ_0.)

$T(A,B,\varphi) \triangleq$ if $T_0(A,B,\varphi')$ then $\varphi = \varphi'$.

STS: $J(T) \subseteq T$ "a pre-fixed point"

$\therefore T_0 \subseteq T$. ↳ To show this, it is sufficient to consider each clause of J.

Today: "Gödelean wheel" renamed to "Gödelian spiral".

Let's revisit formal type theory – ITT:

Inductively defined.

We showed that if $\Gamma \vdash M : A$, then $|\Gamma| \gg |M| \in |A|$

"erasure" / compiled

Called out that $Id_A(M,N)$ is not an adequate notion of equality, though it is a partial equivalence rel'n and has transport / provides indiscernability of identicals. But the empty rel'n also provides this. ~~But~~ You also don't have a notion of function extensionality. The problem is that Id_A is defined uniformly in A.

Indiscernability of identicals:

Idea: families respect identity predicates.

Weakest form: { If $B(M)$ true and $Id_A(M,N)$, then $B(N)$.

predicate (family) — proof of ident

$\Gamma, a : A \vdash B$ type $\qquad \Gamma \vdash P : Id_A(M,N)$

$\Gamma \vdash Q : [M/a]B$ proof of $[M/a]B$

$\Gamma \vdash J[\ldots](\ldots ; P)(Q) : [N/a]B$

Transport

$tr[a.B](refl_A(M))(Q) \equiv Q : [M/a]B$

β-rule / simplification

The trouble arises because, as the identity, it never does anything.

The idea is that P is evidence for the interchangeability of $M, N : A$. The family $a.B$ has an action on such identifications in its index element. To be a valid family, you must build in the property that it respects this. But in ITT, it's not the family that knows this: it's built in to the system generically. And by M-C's thm, the only identification is refl. So the whole setup breaks.

WTS

$$\neg Id_{Nat}(zero, \overbrace{succ(zero)}^{one})\ true,$$

i.e., $p : Id_{Nat}(zero, one) \vdash M : \bot$ true,
where $\bot$ is the nullary sum.
What could M be?
Suppose we had a P s.t.

$$[zero, one/a.b]P \equiv \bot$$
$$[M.M/a,b]\ P \equiv T$$

$\dfrac{M:A \quad A=B}{M=B}$

Take

$$J[a,b,-.P](a.\langle\rangle, p) : [zero, one/ab]P.$$

Use transport:

$$tr[a.P](p : Id_A(zero, one))(\langle\rangle) : \bot$$
$$: \underbrace{[zero/a]P}_{\equiv T} \to \underbrace{[one/a]P}_{\equiv \bot}.$$

Where $[one/a]P \equiv \bot$ and $[zero/a]P \equiv T$.
How do you define P?
(Large elims:)

$NATREC(T; -, - . \perp)(a)$
$\ddot{N}at$

$a =$ zero gives T
$a \neq$ zero gives $\perp$.

Unfortunately, there's no NATREC in ITT. And in fact, you can show that there is no such P in ITT, and you can't show $\neg Id_A(zero, one)$, i.e., $Id_A(zero, one)$ is consistent w/ ITT!

Two moves:

1. Add or allow "large elims"
2. use natrec ... but what is the type of this particular use? Add a "type of types", i.e., a universe $\mathcal{U}$.

$$\frac{M : \mathcal{U}}{M \text{ type}} \qquad \frac{M \equiv M' : \mathcal{U}}{M \equiv M' : \mathcal{U}}$$

But this causes more problems. Cf. Russell's paradox.

M-L's first type theory was inconsistent b/c it had $\mathcal{U} : \mathcal{U}$.

2018-03-22

Last time

We wanted to show that

$\neg Id_{Nat}(zero, one)$ true

The crucial idea was that we had to define a type by induction on natural numbers:

$$\begin{cases} T(a\ nat) \equiv NATREC(T; -, - . \perp) \\ T(zero) \equiv T \\ T(one) \equiv \perp \end{cases}$$

This indicates one thing left out by the formal system (Cf. Gödel).

But analogously, set theory provides no size for the set of reals. The ~~different~~ thing is, set theory has no notion of truth. A formalism will always be incomplete, but at least $\bar{\omega}$ CTT we have a notion of truth that gives meaning / a basis upon which we can judge any new axioms we add.

Continuing from last time:

1. Add it explicitly, but saying b below should be a type is very dangerous:

X $$\frac{\Gamma \vdash A_0 \text{ type} \qquad \Gamma, a : Nat, b : ? \vdash A_1 \text{ type}}{\Gamma \vdash NATREC(A_0, a, b, A_1)(M) : \text{type}}$$

So instead, use

2. Universes: ("maximal") collections of types (or if you really want, of codes for types):

$$\frac{\Gamma \vdash A_0 : \mathcal{U} \qquad \Gamma, a : Nat, b : \mathcal{U} \vdash A_1 : \mathcal{U}}{\Gamma \vdash natrec(A_0, a, b. A_1)(M) : \mathcal{U}}$$

You compute a type out of a number, eliminating into $\mathcal{U}$, the "type of types".

well-defined given that we have $\mathcal{U}$.

Remark: adding universes is a non-conservative extension: it gives rise to more inhabitants of types than you would have had otherwise.

If you want to use codes, you say elements of $\mathcal{U}$ are types ("El" decodes the code A into a type):

$$\frac{\Gamma \vdash A : \mathcal{U}}{\Gamma \vdash A \text{ type}} \qquad \frac{\Gamma \vdash A = B : \mathcal{U}}{\Gamma \vdash A = B} \quad \text{(Elim)}$$

$El(A)$

~~Bad~~ If you do this, you need syntax for the codes for your base types, e.g., $El(\underline{nat}) = Nat$
$El(\underline{bool}) = Bool$, etc.

Continuing w/o codes:

$$\frac{}{\Gamma \vdash Nat : \mathcal{U}} \qquad \frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma \vdash B : \mathcal{U}}{\Gamma \vdash A + B : \mathcal{U}}$$

Bool
Void
$\mathcal{U}_{nat}$

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma, a : A \vdash B : \mathcal{U}}{\Gamma \vdash \Pi a : A . B : \mathcal{U}} \qquad \Sigma a : A . B : \mathcal{U}.$$

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash M, N : A}{\Gamma \vdash Id_A(M, N) : \mathcal{U}.}$$

With universes, we now have the ability to compute types.

Note: You do not have $\mathcal{U} : \mathcal{U}$!

Instead, Oldea: use an infinite cumulative hierarchy of universes

$$\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \cdots$$

$$\frac{\Gamma \vdash M : \mathcal{U}_i}{\Gamma \vdash M : \mathcal{U}_{i+1}} \text{, cumulativity}$$

Big question

Before, Id_A was defined uniformly in A. This was not very nice because you don't have

1) $Id_{Nat \to Nat}(f,g)$ iff $\Pi a : Nat(fa, ga)$ X

2) What is $Id_U(A,B)$? In trying to answer this, V. Voevodsky came up w some answers to 1).

(Remark: The entire question is illegitimate: just because you want Id to be "equality" doesn't make it so.)

One could take $Id_U(A,B)$ to be isomorphism:

$f: A \to B$, $g: B \to A$, w $g \circ f = id_A$, $f \circ g = id_B$.

What are these "$=$"?

But $(g \circ f)(a) = a$ is too fine, you can't really get this in general.

$Id_A(g(f(a)), a)$ is a bit better.

What V. V. came up w is, at a high level, isomorphism up to isomorphism.

Idea: explore the generic structure of identity types.

Previously:

$refl_A(M) \quad Id_A(M,M)$

$sym_A(P) : Id_A(N,M)$ if $P : Id_A(M,N)$

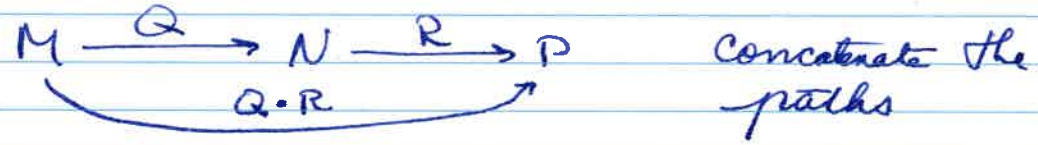
$M \xrightarrow{P} N \rightsquigarrow M \xleftarrow{P^{-1}} N$

$sym_A(P) = rev_A(P) \equiv P^{-1}$

$trans_A(Q,R) : Id_A(M,P)$

if $Q : Id_A(M,N)$

$R : Id_A(N,P)$



Concatenate the paths

$cat_A(Q,R)$ $\quad Q \cdot R$

Properties $(refl_A(M))^{-1} \equiv refl_A(M)$ ("inevitable")

Depending on the code for $trans_A(Q,R)$, we may have

$trans_A(Q, refl_A(N)) \equiv Q$ right ind

Or we may not. It entirely depends on your implementation.

The problem with dependent type theory is that the code goes into the classifier, so modularity goes out the window: you depend on a particular implementation.

You may also have

$trans_A(refl_A(N), R) \equiv R$ left ind

If you do double induction on the paths, then you only get

$trans_A(refl_A(M), refl_A(M)) \equiv refl_A(M)$.

What do you get if you do

$P \cdot P^{-1}$ or $P^{-1} \cdot P$?

$\not\equiv refl_A(M)$

in general. How about

$P \cdot (Q \cdot R)$? $(P \cdot Q) \cdot R$

$(P^{-1})^{-1} \quad ? \quad P$

$P \cdot refl \quad ? \quad P$

$refl \cdot P \quad ? \quad P$

These are called the <u>groupoid laws</u>

The expectation is that these laws should hold for identifications. They do hold!

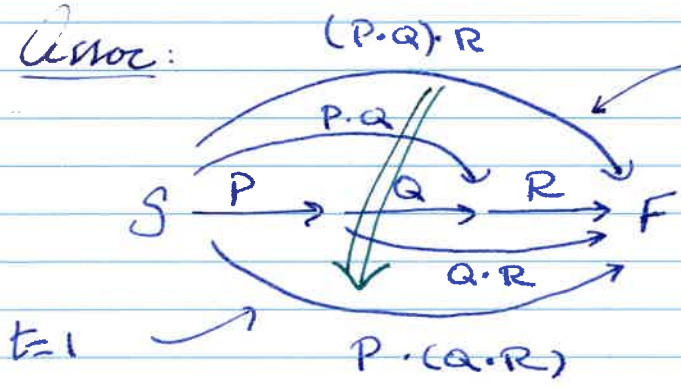
Suppose $P : Id_A(M,N)$ so that $P^{-1} : Id_A(N,M)$.

<u>Claim</u>: $\exists : Id_{Id_A(M,N)}(P \cdot P^{-1}, refl_A(M))$.

↪ Exercise: use J (Id-elim) on P.

The critical idea is this iterated identification: you have identification of identifications. (Yes, we know all closed instances of Id are refl.)

<u>Assoc</u>:



In what sense are the top and bottom paths the same? They certainly aren't ~~the~~ syntactically equal. Instead, what you loosely have is some homotopy / continuous deformation from the top to the bottom. But we're programming so you need some ~~computation~~ comprehension principle for them. The whole point of univalence was to add enough such deformations.

$$\mathcal{J} : \mathrm{Id}_{\mathrm{Id}_A(s,f)}\big((P\cdot q)\cdot R,\ P\cdot(Q\cdot R)\big)$$

↳ Do it using path induction / J.

In short, the groupoid laws hold "up to higher identification". These iterated identifications bring in dimensionality.

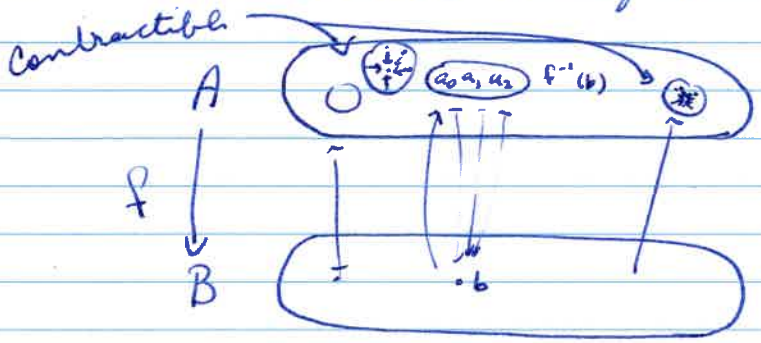
points 0
paths between pts 1
paths btw — 2
paths btw — 3
⋮

The structure you get is that of an ∞-groupoid.

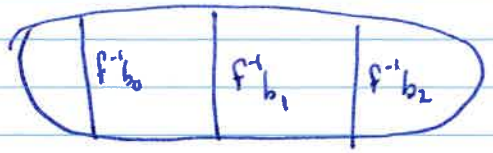
Next

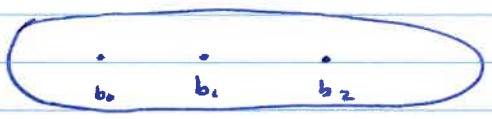
→ Develop the univalence axiom (UV)

The key technical idea is inspired by the following idea:
When is $f : A \to B$ a bijection?



$f^{-1}(b)$ - preimage : $\Sigma a{:}A.\ \mathrm{Id}_B(fa, b)$
"fiber of f over b".





f is a bijection exactly when each of these fibers consists of a single point up to identification, i.e., is contractible.

2018-03-27

Homotopy Type Theory (HoTT)

extension of ITT w/ two ideas:

1. univalence - types are identified up to equivalence (isomorphism up-to iso)
2. (higher) inductive types that specify points and also paths/identifications

Notation

$Id_A(M,N)$ identity type
$M =_A N$ identification type
" path type "

elim form: " path induction "
induction form: $refl_A(M) : M =_A M$

Last time we saw the groupoid laws/structure of idents:

$refl_A(M) : M =_A M$ (prim)
if $P : M =_A M'$, then $P^{-1} : M' =_A M$ (def'ble)
if $P : M =_A M'$ and $Q : M' =_A M''$ then $P \cdot Q : M =_A M''$, (def'ble)

s.t. these are all inhabited:

- $refl_A(M) \cdot Q =_{M =_A M'} Q$
- $P \cdot refl_A(M') =_{M =_A M'} P$
- $refl_A(M)^{-1} =_{M =_A M} refl_A(M)$
- $P^{-1} \cdot P =_{\_\_} refl_A(M')$
- $P \cdot P^{-1} =_{\_\_} refl_A(M)$
- $(P^{-1})^{-1} =_{\_\_} P$
- $P \cdot (Q \cdot R) =_{\_\_} (P \cdot Q) \cdot R$

Remarks:

If $f: A \to B$ and $P : M =_A M'$, then there is an operation

$$ap_f(P) : f(M) =_B f(M')$$ "applied"

(which we will later write as "$f(P)$")

such that

$$ap_f(refl_A(M)) \equiv refl_B(f(M)).$$

Pf: By path induction on P:

$$J[a, b, \ldots f(a) =_B f(b)](a.\, refl_B(f(a)))(P)$$

$$: f(M) =_B f(M') \qquad \square$$

Though HoTT is not computational, it has a notion of computation via proof reduction à la Gentzen.

Recall

$a : A \vdash B$ type $\quad P : M =_A M'$

$tr[a.B](P) : [M/a]B \xrightarrow{\sim} [M'/a]B$

"$refl_A(M)(Q) \equiv id_{[M/a]B}$"

If $f : \Pi a : B.\, B(a)$ and $P : M =_A M'$, then you would like something like

"$f(M) =_{??} f(M')$"

$[M/a]B \qquad [M'/a]B$

The two sides live in different types... Use "heterogeneous paths", say P induces some path $apd_f(P)$:

$$f(M) : [M/a]B \;\; \overset{\text{horizontal}}{\underset{apd_f(P)}{\cdots}} \;\; f(M') : [M'/a]B$$

$$\xrightarrow{tr[a.B](P)(f(M))} \quad \text{vertical } ap_f(P)$$

$$M \xrightarrow{\quad P \quad} M'$$

$apd_f(P) \triangleq$

$$J[a,b,p.\ tr[a.B](p)(f(a)) =_{[b/a]B} f(b)](a.\ refl_B(f(a)))(P)$$

$$: tr[a.B](P)(f(M)) =_{[M'/a]B} f(M')$$

(Also in HoTT book)

"$apd_f(P) : f(M) =^{a.B}_{P} f(M')$"

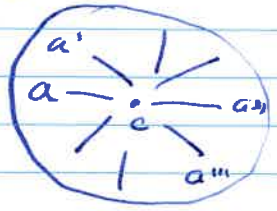
is the notation we use in the dep case.

Singletons aka contractibility express the idea of unique existence (up to identification). Both a boon and a bane: many things are equal and you don't need to worry about differences, but you also don't have fine shades of distinction.

is Singleton(A) aka isContractible(A)

$$\Sigma_{c:A}\ \Pi_{a:A}.\ \underbrace{a =_A c}_{\text{identification}}$$

"centre"



$f : A \to B$ is an equivalence iff "f is a bijection", iff "$\underbrace{f^{-1}(b)}_{\Sigma a:A.\ f(a)=_B b.}$ is a singleton for all $b:B$"

You can then define f^{-1} by sending each b to the ~~centre~~ a that is the centre of the preimage.

What do we mean by "equality of types"?

Idea: "pre-univalence"

$A =_{\mathcal{U}} B$ iff $\underbrace{A \simeq B}_{\Sigma f:A\to B.\ isEquiv(f)}$

$\Pi_{b:B.\ isContr(f^{-1}(b))}$

Univalence gives you more than this.

$A =_{\mathcal{U}} B$ ~~iff~~ $A \simeq B$

A type w/ elements: paths identifying A & B $\simeq$ A type of equivalences between A & B.

Want to establish some equivalence btw these 2 types.

But we can say even more!

Define $\mathrm{idtoEquiv}_{A,B} : \underbrace{A =_{\mathcal{U}} B}_{\mathrm{Id}_{\mathcal{U}}(A,B)} \longrightarrow \underbrace{A \simeq B}_{\mathrm{Equiv}_{\mathcal{U}}(A,B)}$

using path induction:

(At the end of the day, can only be refl)

$\lambda p : A =_{\mathcal{U}} B.\ J[a, b, \_, a \simeq b](a. \langle id_A, \underbrace{\_\_\_}_{\mathrm{Equiv}_{\mathcal{U}}(a,a)} \rangle)(p)$

↑ fibres are contractible

Univalence Axiom:

"idtoEquiv is an equivalence"

$ua(A, B, E) : \mathrm{Id}_{\mathcal{U}}(A, B)$

$E : A \simeq B$

↳ defn depends on the axiom and is irreducible and cannot be reduced to refl!

The upside from a mechanization POV is that you identify types up to equivalence.

This is the start of a theory of symm coercion!

Problem

$J[a, b, p. C](a. Q)(ua(A, B, E)) \equiv$ (?)

How do we "execute" / "simplify" / "compute" w/ univalence? This is a stuck state and as well see, there's no escape short of

rethinking everything.

Idea: express it judgmentally, and then internalize it.

Idea: $\mathrm{tr}\,[X.X]\,(\mathrm{ua}(E))\,(M:A)_{1=B} \equiv$ "$E(M)$" $\equiv$

should come out to be true. And indeed, in the model of the cat of simplicial sets, it does come out to be true.

The only reason to worry about all of these things is that we want computation.

Use "Martin-Löf's martial arts moves where he hardly moves and everybody is on the floor."

(Higher) Inductive Definitions

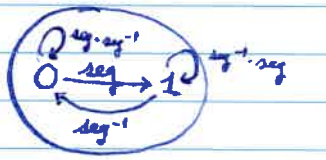
Idea: specify the free type on some generators. In group theory, you have free groups on generators satisfying ~~some~~ rel'ns, but in ~~one~~ TT, you don't have just rel'ns.

Example: Interval type I

$0 : I$ "points"

$1 : I$

$\mathrm{seg} : 0 =_I 1$ "line"



$\Sigma a,b:I.\ \mathrm{Id}_I(a,b)$

So to speak, there's additional stuff brought in by the fact that you're generating the free ∞-groupoid on the generators. The reason to worry about it is that when you map out, you not only need to worry about what to do on 0 and 1 (as you would with bool), and on seg, but also all of the other stuff.

Next time will derive the eliminator & recursor.

2018-03-29

Univalence

Implies that equal types in a universe are exactly equivalent types

$$\mathrm{idtoEquiv} : A =_{\mathcal{U}} B \xrightarrow{\sim} A \simeq B$$

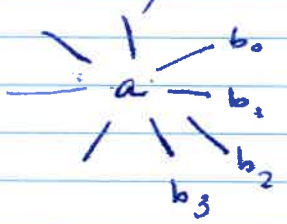
$\Rightarrow$: definable by path induction

$\Leftarrow$: [axiom. a way to build paths in $\mathcal{U}$ out of equivalence (eual(A,B,E) is a "new" identification]

Claim (V.V.) : univalence suffices for function extensionality.

Def (total path space)

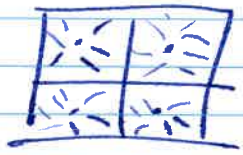
$$\mathrm{Paths}(A) = \Sigma a : A . (\Sigma b : A . a =_A b)$$



"based path space at a"
"star at a"
"equivalence class at a"

Remark that path spaces are contractible singletons.

Fact: $A \simeq \mathrm{Paths}(A)$



Indeed, every elem of A has a path space and every path space contracts to an equiv elem of A

Q(A):

And so by univalence, $\boxed{A =_{\mathcal{U}} \mathrm{Paths}(A)}$.

Remark : J $\Rightarrow$ transport + contr based path spaces.

But also J $\Leftarrow$ ____

See the work by C. Anjuli for details.

Def: $\mathrm{Htpy}(A,B) \triangleq \Sigma f,g : A \to B.\ f \sim g$

where $f \sim g \triangleq \Pi a:A.\ f(a) =_B g(a)$.

Fact: $(A \to \mathrm{Paths}(B)) \simeq \mathrm{Htpy}(A,B)$

Pf: $(\to)$ ~~$\lambda F.$~~ $\lambda F.\ \langle \lambda a:A.\ F(a)\cdot 1,\ \lambda a:A.\ F(a)\cdot 2,\ \lambda a:A.\ F(a)\cdot 3\rangle$

$(\leftarrow)$ $\lambda H.\ \lambda a:A.\ \langle (H\cdot 1)(a), (H\cdot 2)(a), (H\cdot 3)(a)\rangle$.

You can show these are mutually inverse. □

Remark $\mathrm{Paths}(A \to B) \simeq \mathrm{Htpy}(A,B)$ expresses function extensionality.

$\mathrm{Htpy}(A,B) \hookrightarrow \mathrm{Paths}(A \to B)$ is easily definable.

$\hookleftarrow$ is as follows:

$$\left.\begin{aligned} \mathrm{Paths}(A \to B) &=_{\mathcal{U}} A \to B \\ &=_{\mathcal{U}} A \to \mathrm{Paths}(B) \end{aligned}\right\} \text{tr}_{[A \to -]}(Q(B))$$

$$=_{\mathcal{U}} \mathrm{Htpy}(A,B).$$

These are equalities

Cor: Univalence entails function extensionality.

Inductive types

Key idea: inductive types are characterized by a "mapping out" property — initial objects.

eg) natural numbers

$$\frac{}{\Gamma \vdash 0 : \mathbb{N}} \qquad \frac{\Gamma \vdash M : \mathbb{N}}{\Gamma \vdash \mathrm{succ}(M) : \mathbb{N}}$$

I — zero — succ

You say this is the least class by saying if you want to map out, it's suff to consider only 0 and $s(0)$.

$$\frac{\Gamma \vdash M : \mathbb{N}at \qquad \Gamma \vdash A \text{ type} \qquad \Gamma \vdash M_0 : A \qquad \Gamma, a:A \vdash M_{succ} : A}{\Gamma \vdash natrec(M_0 ; a.M_{succ})(M) : A}$$

Dependently if you want:

$$\frac{\Gamma \vdash M : \mathbb{N}at \qquad \Gamma, a:\mathbb{N}at \vdash A \text{ type} \qquad \Gamma \vdash M_0 : A(0) \qquad \Gamma, n:\mathbb{N}at, a:A(n) \vdash M_{succ} : A(succ(n))}{\Gamma \vdash natrec(M_0 ; n,a.M_{succ})(M) : A(M)}$$

(This second elim is definable from the simply-typed one + Σ-types)

$$natrec(M_0 ; n,a.M_{succ})(0) \equiv M_0 : A(0)$$

$$natrec(M_0 ; n,a.M_{succ})(succ(M)) \equiv \equiv [M, natrec(-)(M)/n,a]M_{succ} : A(succ(M))$$

+ Claim / postulate uniqueness up to identifications

"Higher" Inductive Types

idea: A is somehow coupled to $Id_A(-,-)$

$Id_A(-,-)$ —— $Id_{Id_A(-,-)}(-,-)$

⋮

Therefore we can consider a certain form of definition that populates not only the type A but also its companion Id types.

("The beauty of formal type theory is that you can write down whatever.")

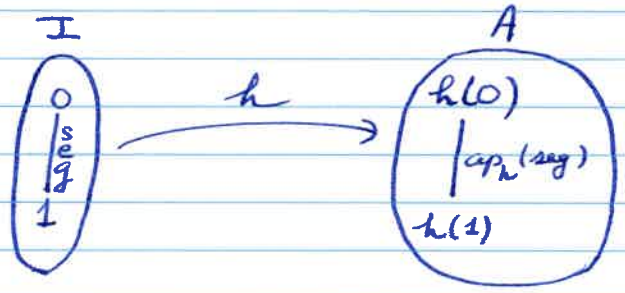
Example "the interval" I, which is inductively defined as follows:

these are the intro & rules & nothing else

$$\frac{\Gamma \vdash 0 : I \qquad \Gamma \vdash 1 : I}{\Gamma \vdash seg : Id_I(0,1)} \quad \text{"points"}$$

$0 \xrightarrow{seg} 1$ **Fact** I is contractible.

Consider some $h : I \to A$



The eliminator is:

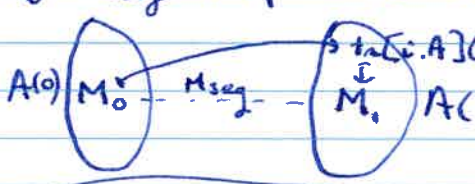
~~$\Gamma \vdash M : I$ $\quad \Gamma, i : I \vdash A$ type~~
~~$\Gamma \vdash M_0 : A(0)$ $\quad \Gamma \vdash M_1 : A(1)$~~

$$\frac{\Gamma \vdash M : I \qquad \Gamma \vdash A \text{ type} \qquad \Gamma \vdash M_0 : A \qquad \Gamma \vdash M_1 : A \qquad \Gamma \vdash M_{seg} : Id_A(M_0, M_1)}{\Gamma \vdash I_{rec}(M_0, M_1, M_{seg})(M) : A}$$

$I_{rec}(-)(0) \equiv M_0$
$I_{rec}(-)(1) \equiv M_1$
$ap(I_{rec}(-)(-))(seg) =_{(M_0 =_A M_1)} M_{seg}$
↳ no computation here

$$\frac{\Gamma \vdash M : I \qquad \Gamma, i : I \vdash A \text{ type} \qquad \Gamma \vdash M_0 : A(0) \qquad \Gamma \vdash M_1 : A(1) \qquad \Gamma \vdash M_{seg} : M_0 =^{i.A}_{seg} M_1}{\Gamma \vdash I_{rec}(M_0, M_1, M_{seg})(M) : A(M)}$$

where $M_0 =^{i.A}_{seg} M_1$ means (by def'n)



$tr[i.A](seg)(M_0) =_{A(1)} M_1$.

$0 \xrightarrow{seg} 1$ ∎

and $\mathrm{Irec}(\_\_)(0) \equiv M_0$
$\mathrm{Irec}(\_\_)(1) \equiv M_1$
} "computational"

$\mathrm{ap}\,(\mathrm{Irec}(-)(-))'(\mathrm{seg}) = \underbrace{M_0 =_{\mathrm{seg}} M_1}_{\text{heterogeneous path type}}$ } equations / comp

Fact $\mathrm{Paths}(A) \simeq_u \sum_{a,a':A} (I \to A)$

Pf: $\lambda \langle a, a', p\rangle.\ \lambda i : I.\ \mathrm{Irec}[A](a, a', p)(i)$

$\lambda h : I \to A.\ \langle h(0), h(1), \mathrm{ap}_h(\mathrm{seg})\rangle$

+ check these are mutually inverse. □

$\mathrm{Paths}(A \to B) \simeq_u I \to (A \to B)$

$\simeq_u (I \times A) \to B$

$\simeq_u (A \times I) \to B$

$\simeq_u A \to (I \to B)$

$\simeq_u A \to \mathrm{Paths}(B)$

$\simeq_u \mathrm{Htpy}(A, B)$.

So the interval also gives you function extensionality.

Rmk: This is the standard textbook version of homotopy.

2018-04-03

Last Time

"Higher" inductive def'ns — idea is to simultaneously define (by specifying generators) types A, Id_A, Id_{Id_A}, ...

Motivation: A is "coupled" with its path spaces.

1) We introduced the abstract interval I. $0, 1 : I$ (end pts) $seg : Id_I(0,1)$ (line path)

"draws a line in A"

$I^n \longrightarrow A$ n-cube in A for $n \geq 0$

Eliminations for I.

$$\frac{\Gamma \vdash M : I \qquad \Gamma, i : I \vdash C \text{ type} \qquad \Gamma \vdash M_0 : [0/i]C \qquad \Gamma \vdash M_1 : [1/i]C \qquad \Gamma \vdash M_{seg} : M_0 =^{i.C}_{seg} M_1}{\Gamma \vdash I\text{-ind}[i.C](M_0; M_1; M_{seg})(M) : [M/i]C}$$

($M_{seg} : tr[i.C](seg)M_0 = M_1$)

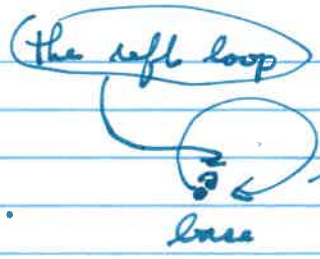
1) $I\text{-ind}[i.C](M_0; M_1; M_{seg})(0) \equiv M_0 : [0/i]C$
$(1) \equiv M_1 : [1/i]C$.

2) $apd(I\text{-ind}[i.C](M_0; M_1; M_{seg})(-))(seg) =_{M_0 =^{i.C}_{seg} M_1} M_{seg}$

↳ This is a bit strange because you're neither here nor there, because you have a notion of computation but you don't at the same time.

Also a bit weird that some of this is ≡/definitional and the other is data for a type.

Circle $\mathbb{C}$ aka S^1



base : $\mathbb{C}$

loop : $\mathrm{Id}_{\mathbb{C}}(\mathrm{base}, \mathrm{base})$.

$$\frac{\Gamma, c : \mathbb{C} \vdash C \ \mathrm{type} \qquad \Gamma \vdash M : \mathbb{C} \qquad \Gamma \vdash M_{\mathrm{base}} : [\mathrm{base}/c]C \qquad \Gamma \vdash M_{\mathrm{loop}} : M_{\mathrm{base}} =^{c.C}_{\mathrm{loop}} M_{\mathrm{base}}}{\Gamma \vdash \mathbb{C}\text{-ind}[c.C](M_{\mathrm{base}}; M_{\mathrm{loop}})(M) : [M/c]C}$$

$$\circledast \quad \mathrm{ap}_{[c.C](\mathrm{loop})}(M_{\mathrm{base}}) = M_{\mathrm{base}}$$

(where the left side has type $[\mathrm{base}/c]C$ and the right side has type $[\mathrm{base}/c]C$)

You could have written $\Gamma \vdash M_{\mathrm{loop}} : M_{\mathrm{base}} =_{[\mathrm{base}/c]C} M_{\mathrm{base}}$ or $\circledast$. It would have type-checked, but would be wrong because it doesn't respect the loop in any real sense: loop isn't brought to a "loop" in C.

There's the loop space

$$\Omega(A; a_0) \triangleq \underbrace{a_0 =_A a_0}_{\mathrm{Id}_A(a_0, a_0)}$$

(base point in A)

in A. The important thing is that the groupoid laws become group laws (where the group structure is up-to higher identification) because the endpoints are the same.

If you want to ensure this is actually a group, you take the "zero-truncation" $\|\Omega(A; a_0)\|_0$, which squashes down all of the higher-dimensions.

The truncated loop space of $\mathbb{C} = (\mathbb{Z}, +, 0)$ → requires univalence to get the equality, but it avoids mucking about with equivalences.

Q: What is the computational meaning of "HoTT"?

Two ways of looking at it
1. constructivism
2. prog lang type systems (coercion, aka homotopy)

Cf. Martin-Löf's papers on judgments (epistemics for type theory).

Idea: Judgments come first — before connectives and type constructors.

Gentzen's key contribution was the stress on entailment (hypothetical judgment) as prior to implication.
Cf. Hilbert systems which have only implication
Cf. λ vs. combinators

The entire idea is $\dfrac{A \text{ true} \vdash B \text{ true}}{A \supset B \text{ true}}$ (entailment) (internalization of ")

This is a "ninja move".

The generalization of entailment is something similar for universal quantification

$$\frac{\vdash_x (A \text{ true})}{\forall x A \text{ true}}$$ (generality) (internalization)

Constructivism (the idea that the evidence for the truth of a prop is a thing) suggests consolidation:
"hypothetico-general judgments"

$$\underbrace{x_1 : A_1, \text{---}, x_n : A_n}_{\text{variables}} \vdash M : A$$

This emphasizes the centrality of variables and what they mean in type theory:

1) formal: indeterminates "derivability" $\Gamma \vdash M : A$, $\Gamma \vdash M \equiv N : A$ $\Longrightarrow$ semantic: placeholders "admissibility" $\Gamma \gg M : A$, $\Gamma \gg M = N \in A$

They both share the structural properties and present a consequence rel'n.

(compositio) Substitutio / transitivity

$$\frac{\Gamma, x : A \gg N : B \qquad \Gamma \gg M : A}{\Gamma \gg [M/x]N : B}$$

(id) reflexivity / variable

weakening / "extra" variables

contraction / "duplicate" variables.

Aside: The first version of Church's λ-calc required you to use all vars. so you couldn't write $\lambda x.\lambda y.x$. This was the λi (λI?) calc. He then changed his mind and then you get the λK calculus (rmk: $K = \lambda x.\lambda y.x$).

Back to the idea that judgments come first:

Judgmental Structure of Identification Paths.

Idea: want to think about paths as an intrinsic concept in type theory.

[You can't just chuck in axioms too TT, because it destroys

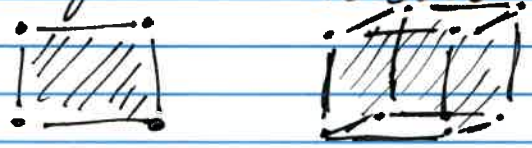
the computational content of it].

Introduce some way of talking about paths prior to any connectives. Multiple ways of doing it, but the world has settled on a cubical structure of dimensions. The dimension in HoTT corresponds to the number of iterations of Id.

A^0 Id_A Id^2_A Id^3_A ...

We want to give these names:

points lines squares cubes ...



We want judgments that capture these.

Quine: "No entity without identity"

$M \doteq M' \in A$ @ 0 (points)

$M \doteq M' \in A$ @ 1 lines. ?? What is it.

To be able to answer this, we must first answer what of it. ~~this~~

$A \doteq A'$ type @ 0 familiar

$A \doteq A'$ type @ 1 unfamiliar.

The thing to recall is that families of types $(F_a)_{a \in A}$ are central to DTT.

$F[M/a]$ instance

The point is that lines in A induce lines between instances.

$M \text{ —— } M' \rightsquigarrow F[M] \text{ —— } F[M']$

abstract identification

coercions transport.

The thing you care about at the end of the day is how the classifiers work. The line is reversible, so you can go back. What if you go there

and back? Developing a symmetric theory of coercion is hard work and what we're after. Univalence turns equivalences between types into lines between types.

So

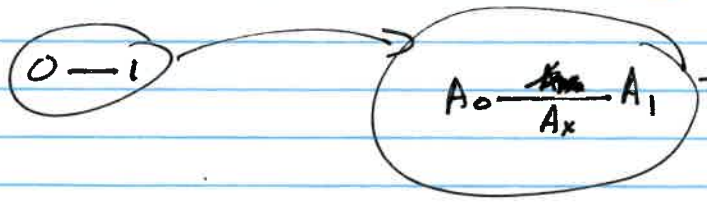
$M \doteq M' \in A$ @ 1 is lines in lines / ~~2~~ heterogeneous path.

Back to the centrality of vars.

Idea: Else Cartesian coordinates. Specify the dimensionality by contexts of variables that "range over" an interval

$A' = A_x$ type $\overbrace{[x_1, \ldots x_n]}^{\Phi}$ $(n \geq 0)$ / $n=0$ is the case we're used to.

$M' = M_x$ type $[x_1, \ldots, x_n]$



"$x_1 \ldots x_n \vdash A$" suggestively means "$x_1 : I \ldots x_n : I \vdash A$" c.e., "$x : I^n \vdash A$ type"

So you get n-cubes.

There's no notion of distance: you're either one of the endpts $\langle 0/x \rangle$ $\langle 1/x \rangle$ but never "half-way" along the interval. Oddly enough, just like "plugging in an endpoint" causes the code to run, so can ~~does~~ plugging in a variable $\langle y/x \rangle$.

The lines in our setup will tell us how to how to coerce.

2018-04-05

Dimension ctxts & dim substs

$\Psi = x_1, \ldots, x_n \quad (n \geq 0)$

dbe "cartesian coords in n-space"

Let $\psi : \Psi' \longrightarrow \Psi$ be a subst for dim vars. If $\Psi = x_1 \ldots x_n$, then $\Psi' \vdash \psi(x_i)$ dbm

$\Gamma ::= 0 \mid 1 \mid x$

Dimension contexts are structural

$\Psi, x \xrightarrow{\text{(silent)}} \Psi$ weakening

$\Psi, z \xrightarrow{\langle 2,2 | x,y \rangle} \Psi, x, y$ contraction

$\Psi, x, y, \Psi' \xrightarrow{\text{silent}} \Psi, y, x, \Psi'$ exchange

M term w dim vars Ψ

If M tm $[\Psi]$ and $\psi : \Psi' \longrightarrow \Psi$ then $M\langle\psi\rangle$ tm $[\Psi']$.

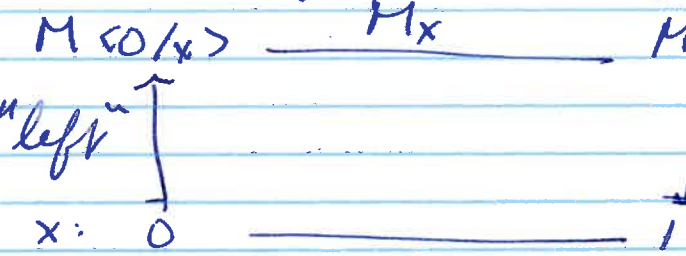
: variant functorial action of ψ on terms
"pullback"

This preserves identity and composition "on the nose"

$$M\langle id \rangle = M$$
$$M\langle \psi_1 \circ \psi_2 \rangle = (M \cdot \psi_1) \cdot \psi_2$$

Q Can give a presheaf semantics
— Kripke semantics where worlds = dimensions

Substitution of 0, 1 picks out "ends"



We have a <u>cubical programming lang</u>

$$\left.\begin{array}{l} M\ \mathrm{val}_{\mathcal{I}} \\ M \longmapsto_{\mathcal{I}} M' \end{array}\right\} M \Vdash^{\mathcal{I}} M'$$

$$\longmapsto_{\Psi} \in \mathrm{Tm}[\mathcal{I}] \times \mathrm{Tm}[\Psi]$$

Could omit the dimension ctxts $\mathcal{I}$ if wanted.

Note, these rules do not commute with subst in general!

$M\ \mathrm{val}_x$ but not $M\langle 0/x\rangle\ \mathrm{val}_0$

$M \longmapsto_x M'$ but not $M\langle 0/x\rangle \longmapsto_0 M'\langle 0/x\rangle$.

eg) circle $\mathbb{C}$ in computational HT.

base val

loop_r

loop_x val

$\mathrm{loop}_0 \longmapsto$ base

$\mathrm{loop}_1 \longmapsto$ base

In some $\mathrm{Ctx}[\Psi]$:

$\mathbb{C}\text{-elim}[c.C](M_{\mathrm{base}}; x.M_{\mathrm{loop}}x)(M)$ of M : M'

$\mapsto$ $\mathbb{C}$-elim ——— (M')

Ψ ——— (base) $\mapsto M_{\mathrm{base}}$

Ψ, x ——— $(\mathrm{loop}_x) \mapsto M_{\mathrm{loop}}$

be careful, x is x in M_{loop} here

Remark that these reductions work at all dimensions.

Note: substituting into a loop changes a value into a non-value!

$loop_x$ val, but $loop_0 \mapsto$ base!

Also note that reduction doesn't commute w/ subst!

$\mathcal{C}\text{-elim}(\longrightarrow)(loop_x)\langle 0/x\rangle = \not\rightarrow(loop_0)$
$\mapsto \not\rightarrow(base)$
$\mapsto M_{base}$

but $\mathcal{C}\text{-elim}(\longrightarrow)(loop_x) \mapsto M_{loop}$

and if you then do the subst, then $\not\equiv M_{loop}\langle 0/x\rangle$?

} diverges.

You have a semantic confluence, but not a syntactic confluence, and you cannot get a coherence via the op sem. The equality is a semantic notion via the type.

What does the judgment $M \in A[\mathcal{I}]$ mean?

(Informal account first)

w/o $\mathcal{I}$, it means $A \Downarrow A_0$ and $M \Downarrow M_0$ } + when are they equal?
and "M_0 is a value of type A_0" }

(It's all about progs running and specifying their values)

A first Desideratum #1, cut w $\mathcal{I}$:

$A \Downarrow^{\mathcal{I}} A_0$, $M \Downarrow^{\mathcal{I}} M_0$, "$M_0$ is a $\mathcal{I}$-value of A_0"

$\Rightarrow$ Need to explain what the $\mathcal{I}$-values are.

But consider: ~~[illegible]~~

if $loop_x \in \mathcal{C}[x]$ then $loop_0 \in \mathcal{C}[\cdot]$?

You want this to be true.

base $\xrightarrow{loop_x}$ loop But you need to make it so.
I I
loops loops.

The meaning of must account for face maps (~~stab to get "sides"~~)

Desideratum #2: (generalizes #1)

If $A \Vdash A_0$, then for all $\varphi : \mathcal{I}' \to \mathcal{I}$ ("$\mathcal{I}$-aspects")

$$M\varphi \in A_0[\mathcal{I}']$$

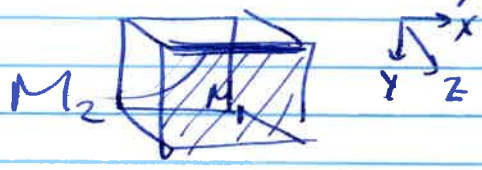
i.e., $M\varphi \Vdash M_0$ and M_0 is a $\mathcal{I}'$-value of A_0.

eg: $loop_x \in \mathbb{C}[x]$ implies

$loop_0 \in \mathbb{C}[\cdot]$
$loop_1 \in \mathbb{C}[\cdot]$

} Should be (and is) valid.

Issue: Suppose you have a cube



$M\langle 1/z\rangle \Vdash M_1$ versus $M\langle 1,0/z,y\rangle \Vdash M_3$
$M_1\langle 0/y\rangle \Vdash M_2$

Is $M_2 = M_3$?

Want to demand that they be exactly equal in whatever type they're in, i.e.,

$$M_2 \doteq M_3 \in A_0[x] \qquad (*)$$

Nothing forces this, could write down bad programs. But: lots of ~~untyped~~ "bad" programs exist and you just decide not to give them a type. Your types are expected to be specifications of "good" programs, so you just design your type system to ensure (*) holds.

Desideratum #3: COHERENT ASPECTS

If $M \in A[\Psi]$ then
$A \Downarrow A_0$, $M \Downarrow M_0$
if $\psi_1 : \Psi_1 \to \Psi_2$ $\Psi : \Psi_2 \to \Psi_3$
$\{ M\psi_1 \Downarrow M_1,\ M_1 \Psi_2 \Downarrow M_2,\ M(\psi_1 \Psi_2) \Downarrow M_{12}$
then $M_{12} \doteq M_2 \in A_0[\Psi_2]$.

This is still not enough! What about DEPENDENCY?

→ types can vary in a dimension!

F A-indexed family of types
F[M], where $M \in A[\Psi]$ is a type
varies in the dimension!

$F[M]\langle o/x \rangle = F[M\langle o/x \rangle]$ (later: other intrinsic ways this can arise)

↳ Dep Des #1:

$\forall\ \psi : \Psi' \longrightarrow \Psi$

$M\psi \in A\psi[\Psi']$

i.e., $M\psi \Downarrow M_0$ and M_0 is a Ψ' value
$A\psi \Downarrow A_0$ of A_0.

The picture for coherent aspects becomes

the front face of M should inhabit the ——— A
the top right edge of M ——— A etc.

It all has to cohere.

Dep. Des #2: Coherent aspects in higher dimensions

If $M \in A(\mathcal{I})$ then

$A \Vdash A_0 \quad M \Vdash M_0$

if $\begin{cases} \Psi_1 : \mathcal{I}_1 \to \mathcal{I}_2 \quad \Psi_2 : \mathcal{I}_2 \to \mathcal{I} \\ M\Psi_1 \Vdash M_1, \quad M_1\Psi_2 \Vdash M_2 \\ A\Psi_1 \Vdash A_1, \quad A_1\Psi_2 \Vdash A_2 \\ M(\Psi_1\Psi_2) \Vdash M_{12} \\ A(\Psi_1\Psi_2) \Vdash A_{12} \end{cases}$

then $M_{12} \doteq M_2 \in A_2[\mathcal{I}_2]$

and $A_{12} \doteq A_2$ type.

Aside. Equality is prior to paths. Because you can only "fit together" to form cubes/squares if you have a notion of equality of endpoints.

Hypothetico-general judgments

What's the meaning of

$a : A \gg N \in B[\mathcal{I}]$?

Want this to be a mapping, so you could say:

if $M \doteq M' \in A[\mathcal{I}]$, then $N[M/a] \doteq N[M'/a]$
$\in B[M/a] \doteq B[M'/a]$ $[\mathcal{I}]$.

But this is not enough, you need something stronger!

You need more than just taking equal things to equal things!

You need to ensure it work not just at the current dimension but at all accessible dims so that you respect paths. "

So you actually want:

$$\forall\, \tau : \mathbb{I}' \to \mathbb{I}$$

if $M \doteq M' \in A\tau$ then

$$N\psi[M/a] \doteq N\psi[M'/a] \in B\psi[M/a] \doteq B\psi[M'/a][\psi']$$

So the "apd" behaviour from HoTT is built-in!

$\mathbb{I}'$ could be $\mathbb{I}, \times$.

" So if you have a line between $\mathbb{I}$-cubes of type A, you get one between $\mathbb{I}$-cubes of type B. "

In summary: make everything behave properly as prescribed.

2018-04-10

Hypothetical Judgments

1) .. $(a : A \gg B \text{ Type})\ [\Psi]$ (families)
2) .. $(a : A \gg N \in B)[\Psi]$ (maps).

~~###~~ Naively (incorrectly), these are "separate facts" at each Ψ.

1) if $M \doteq M' \in A[\Psi]$ then $B[M/a] \doteq B[M'/a]\ [\Psi]$
2) if ——— then $N[M/a] \doteq N[M'/a] \in B[M/a]\ [\Psi]$
$(\doteq B[M'/a])$

The problem is that this isn't strong enough. You want your mapping (captured by the judgment) to be uniform in a sense (natural / polymorphic / etc.) Motivated ~~by~~ by a Kripke semantics point of view; ~~If~~ ~~and~~ you observe that the above is insufficient to maintain truth at the relevant future worlds. More precisely:

1) for all $\Psi : \Psi' \longrightarrow \Psi$ "aspects" (faces, diagonals, degeneracies)

if $M \doteq M' \in A\Psi\ [\Psi']$, then
$N\Psi[M/a] \doteq N\Psi[M'/a] \in B\underset{[M/a]}{\Psi,}\ [\Psi']$.

Our possible worlds are the Ψ and the reachability relation is given by the aspects.

In particular, consider what happens when Ψ is a ~~subst~~ weakening $\Psi : \Psi, x \longrightarrow \Psi$.

{ if $M_x \doteq M'_x \in A\ [\Psi, x]$,
{ then $N[M/a] \doteq N[M'/a] \in B[M/a][\Psi, x]$

i.e., N must take lines to lines and respect paths. A, N and B do ~~might~~ not involve x. But ~~the~~ M

and M' might (and so be lines), so N takes lines equal lines M and M' in A to equal lines in B.

This generalizes to any number of variables.

Crucially

1) Dimension variables behave like ~~independs~~ indeterminates replaced by dimensions ie vars. They do not range over closed terms. This is ~~similar as to~~ variables in formal type theory, and they are given a presheaf semantics.

2) Term variables behave like placeholders for closed terms, just as variables in CTT.

So, in the higher-dimensional case, we combine both types and their techniques.

Note: $M \in A[\mathbb{I}, x] \not\Rightarrow M\langle 0/x\rangle \in A\langle 0/x\rangle$, $M\langle 1/x\rangle \in A\langle 1/x\rangle$

There is more to a line than just it's endpoints.

Let's define some types

(Warning: This will be both boring and wrong.)

eg $Bool \doteq Bool[\mathbb{I}]$ (at any dimension $\mathbb{I}$)

Bool is a degenerate cube on $\mathbb{I}$ (doesn't depend on $\mathbb{I}$).

$true = true \in Bool\,[\mathcal{I}]$
$false = false \in Bool\,[\mathcal{I}]$

We're informally working in the fixed point that defines the types, in a constructor that generalizes the one we saw earlier.

$\overline{true\ val} \quad \overline{false\ val} \quad \overline{Bool\ val}$

$$\frac{M \longmapsto M'}{if[a.C](M; M_0; M_1) \longmapsto if[a.C](M'; M_0; M_1)}$$

$$\frac{}{if[a.C](true; M_0; M_1) \mapsto M_0}$$

$$\frac{}{if[a.C](false; M_0; M_1) \mapsto M_1}$$

Assuming these transitions are stable under dim. subst. This doesn't always hold.

Fact: If ~~$a \in Bool$~~ $a : Bool \gg C\ type\ [\mathcal{I}]$
and $M \in Bool\,[\mathcal{I}]$
and $M_0 \in C[true/a]\,[\mathcal{I}]$
$M_1 \in C[false/a]\,[\mathcal{I}]$
then $if[a.C](M; M_0; M_1) \in C[M/a]\,[\mathcal{I}]$

Rmk: These are coherent instantiations

Proof is similar as before.
(Mar is similar)

$a : A \to B\ type\ [\mathcal{I}]$ iff
$A\ Type\ [\mathcal{I}]$
$a : A \gg B\ type\ [\mathcal{I}]$

Note the apparent circularity. This is not a problem because the construction uses a monot operator and we're working in its fixed point.

(*) $\lambda a.M \in a : A \to B\,[\mathcal{I}]$ iff (def)
$a : A \gg M \in B\,[\mathcal{I}]$

Fact. If $F \in a : A \to B\,[\mathcal{I}]$ and $M \in A\,[\mathcal{I}]$,
then $F(M) \in B[M/a]\,[\mathcal{I}]$.

Note. We're not writing these things down in inference rules because we want to make it clear that we're not

~~defn~~ ~~inductive~~ inductively defining some formal objects.

Remark: Because functions internalize mappings (see ⊗), it is sufficient to get mappings right to get functions right.

In HoTT, we needed apd. Here, we don't, because there's an intrinsic notion of action on path built-in to the system.

$Eq_A(M,N)$ type $[\mathcal{I}]$ — could write this type as "$M =_A N$". Do not confuse as a judgment. (intuition: internalize equality)

iff (def'n) A type $[\mathcal{I}]$, $M \in A[\mathcal{I}]$, $N \in A[\mathcal{I}]$

$P \in Eq_A(M,N)[\mathcal{I}]$

(†) iff $P \Downarrow refl(R)$, $M \doteq R \doteq N \in A[\mathcal{I}]$ } inductive def'n

Notice "equality reflection": if $P \in Eq_A(M,N)[\mathcal{I}]$ then $M \doteq N \in A[\mathcal{I}]$. — Nothing really to say: obvious by def'n.

It internalizes exact equality.

Fact. $Eq_A(-,-)$ is the least reflexive rel'n

This can be internalized as an elim form, giving you the J you had before.

Define

$$\begin{cases} J[a,b,c.C](a.Q)(P) \mapsto \ldots (P') \text{ if } P \mapsto P' \\ \quad \text{refl}(R) \mapsto [R/a]Q \end{cases}$$

<u>Fact</u> If $a:A, b:A, c:Eq_A(a,b) \gg C$ type $[\mathcal{I}]$
$a:A \gg Q \in C[a,a,\text{refl}(a)]$ $[\mathcal{I}]$
$P \in Eq_A(M,N)$ $[\mathcal{I}]$

then $J[a,b,c.C](a.Q)(P) \in C[M,N,P]$.

This says $Eq_A(-,-)$ is the least refl rel'n: If you cover all of the refl cases then you know what to do in all cases, i.e., the refl cases cover all cases.

Fix M. ⓘ $\in C[(M,(N,P))]$ $[\mathcal{I}]$
$Q[M/a] \in C[M,\underline{(M,\text{refl}(M))}]$ $[\mathcal{I}]$

Want to see the underlined pairs as equal members of

$a:A \times Eq_A(M,a)$

$(M,\text{refl}(M)) \in$ " $[\mathcal{I}]$
$\doteq$
$(N,P) \in$ " $[\mathcal{I}]$

Because of equality refl: $M \doteq N$ by P. So $\text{refl}(M) \doteq P$ by (‡). So the pairs are equal. So ⓘ is $Q[M/a]$.

One can make the whole same arg w/ paths.

<u>Critical</u>: $P \in Eq_A(M,N)$ iff $P \Downarrow \text{refl}_A(R)$ s.t. —

(Historically, Eq & Id were confused because of this fact.)

Path type

Idea: internalize the path structure and go back & forth between paths and points, similarly to how we internalized the mapping structure as functions.

$\mathrm{Path}_{x.A}(M,N)$ type $[\mathcal{F}]$

iff A type $[\mathcal{F}, x]$

(def) $M \in A\langle 0/x\rangle$

$N \in A\langle 1/x\rangle$.

$P \in \mathrm{Path}_{x.A}(M,N)\ [\mathcal{F}]$ iff (def)

$P \Downarrow \langle x.Q\rangle$ ("quoting" a line)

$Q \in A\ [\mathcal{F}, x]$

$Q\langle 0/x\rangle \doteq M \in A\langle 0/x\rangle [\mathcal{F}]$

$Q\langle 1/x\rangle \doteq N \in A\langle 1/x\rangle [\mathcal{F}]$

Here, paths wear their endpoints on their sleeve. You could alternatively have a "Path" type as in RedPRL that doesn't mention them.

Define $M@r \longmapsto M'@r$ when $M \longmapsto M'$

$\langle x.Q\rangle @ r \longmapsto Q\langle r/x\rangle$

Fact: If $P \in \mathrm{Path}_{x.A}(M,N)$ then

$P@0 \doteq M \in A\langle 0/x\rangle$

$P@1 \doteq N \in A\langle 1/x\rangle$

2018-04-12

Last time

We saw a cubical semantics for dtt:

$\Gamma >> A = B \ [\mathcal{I}]$

$\Gamma >> M = N \in A \ [\mathcal{I}]$

satisfying a presheaf condition

if $\mathcal{J}[\mathcal{I}]$ and $\phi : \mathcal{I}' \to \mathcal{I}$, then $\mathcal{J}'$.

In particular, degeneracies give rise to a path preserving condition.

Standard types have standard explanations via a fixed point construction.

Note: pre-sheaf cond ensures that functions act on args from all accessible worlds.

- $Eq_A(M,N)$ type $[\mathcal{I}]$ iff A type $[\mathcal{I}]$, $M, N \in A[\mathcal{I}]$

  $refl(R) \in Eq_A(M,N)[\mathcal{I}]$ iff $M = R = N \in A[\mathcal{I}]$.

  *

  This ends up (as we'll see) only being a "pre-type". No need for an elim form, b/c $P \in Eq_A(M,N)[\mathcal{I}]$ iff $M = N \in A[\mathcal{I}]$.

- $Path_{x.A}(M,N)$ type $[\mathcal{I}]$ iff

  A type $[\mathcal{I}, x]$, $M \in A\langle 0/x\rangle \ [\mathcal{I}]$, $N \in A\langle 1/x\rangle \ [\mathcal{I}]$

  $\langle x.P\rangle \in Path_{x.A}(M,N)$ iff

  $P \in A[\mathcal{I}, x]$

  $P\langle 0/x\rangle = M \in A\langle 0/x\rangle \ [\mathcal{I}]$

  $P\langle 1/x\rangle = N \in A\langle 1/x\rangle \ [\mathcal{I}]$

Both of these are inductive definitions

When $x.A$ is degenerate, then this looks a lot like $Id_A(M,N)$ in HoTT (homogeneous lines) $= Path_{\_.A}(M,N)$.

Define $M@r$ as follows:

$$\frac{M \mapsto M'}{M@r \mapsto M'@r} \qquad \frac{}{\langle x.P\rangle @ r \mapsto P\langle r/x\rangle}$$

Fact: If $P \in Path_{x.A}(M,N)[\mathcal{F}]$, then

$$P@x \in A_x[\mathcal{F}, x]$$
$$P@0 \doteq M \in A\langle 0/x\rangle[\mathcal{F}]$$
$$P@1 \doteq N \in A\langle 1/x\rangle[\mathcal{F}]$$

Though $Eq_A(M,N)$ has no elim form, J is "definable" for Eq_A

$$J[a,b,c.C](a.Q)(refl(M)) \mapsto Q[M/a].$$

Under suitable conditions we've seen before, we have

1) $Q[M/a] \in C[M,M,refl(M)/a,b,c]$
2) b/c $P \in Eq_A(M,N)[\mathcal{F}]$, we know
   a) $P \doteq refl_A(M) \in Eq_A(M,N)[\mathcal{F}]$
   b) $M \doteq N \in A[\mathcal{F}]$
3) We can "transport" $Q[M/a]$ from $C[M,M,refl(M)]$ to $C[M,N,P]$: by a remark from last time, these two types are equal/the same.

Fact: $Path_{-.A}(-,-)$ admits a J too, but it will not compute on degeneracies!

Pf: Need to know about Kan operations, introduced below. □

(Kan ops needed to define transport & to build a line ~~between~~ from $C[M,M,refl(M)]$ — $C[N,N,P]$.

(Understanding this fact was a very important development.)

Kan conditions arise from the following:

1. Of what use is a path?
   → paths between types are what matter — they induce coercions ⇒ equivalences and the only use of paths within types

is to define paths between types.

2. What paths are there?
   - univalence
   - paths induced by families
   - Closure conditions

→ We saw the following faux def'n of $\mathcal{C}$:

C type $[\Psi]$
base $\in \mathcal{C}[\Psi]$
$loop_x \in C[\Psi, x]$

⇝ base $\xrightarrow{loop_x}$ base

$loop_0 \mapsto$ base $loop_1 \mapsto$ base

$\mathcal{C}\text{-elim}[c.C](M_{base}, x.M_{loop})(base) \mapsto M_{base}$
$(loop_x) \mapsto M_{loop}$

This runs!

This works and you can prove various things to it, but it isn't what you want: there are no elements $loop_x \cdot loop_x$ or $loop_x^{-1}$ capturing going around the loop multiple times or in the opposite direction. You want this to be the free type on these generators, ~~what~~ because this is inductively defined. Will see what we mean by this later, but these are the closure conditions.

Will see a nice interplay between the pos & neg types: the negative types will already have all of the closure structure req'd; pos type will need to be augmented.

Every inductive type (including C) is positive

Idea

Lines represent "identifications" of their end points (it is misleading to call them equal). Identifications are witnesses to interchangeability.

$$M_0 \xrightarrow{M_x} M_1$$

1) They should be sym & trans if they are to be thought of as identifications.

$$M_1 \xrightarrow{M_x^{-1}} M_0 \ ?$$

$$M_0 \xrightarrow{M_x} M_1 \doteq N_0 \xrightarrow{N_x} N_1$$

adjacent

(~~Your~~ Equality must be prior to identification, because if your lines are "equality", then how do you know they have the same endpoint? Using another line? Repeat ad infinitum.)

$$M_0 \xrightarrow{M_x \cdot N_x} N_1$$

2) Substitutive — coercion prop.

If $M_0 \xrightarrow{M_x} M_1$ and F is a family, then you want

$$F[M_0] \xrightarrow{F[M_x]} F[M_1].$$

→ Witness interchangeability.

Question

If $p \in Path_{-.A}(M,N)\,[\Gamma]$
$q \in Path_{-.A}(N,P)\,[\Gamma]$,
then
$?\ p \cdot q \in Path_{-.A}(M,P)\,[\Gamma]$?

Define $F(-) \doteq Path_A(M,-)$.

↳ an A-indexed family of types:

$a : A \gg F(a)\ type\ [\Gamma]$

(Need to use the action of F on paths)

— A is constant / homogeneous case.

Notice $q@x \in A\,[\Gamma, x]$ ↖
$q@0 \doteq N \in A\,[\Gamma]$ the important thing is
$q@1 \doteq P \in A\,[\Gamma]$ that we have this line.

Looking at the presheaf condition:

The

$F[q@x]\ type\ [\Gamma, x]$ is a line.

And ~~$F[q]$~~ $_{fx}$

$F[q@x]\langle 0/x\rangle = F[q@0] \doteq F[N]$
$F[q@x]\langle 1/x\rangle = F[q@1] \doteq F[P]$

$$F[N] \xrightarrow{\quad F[q@x] \quad} F[P] \qquad \xrightarrow{x}$$

$F[N]$ = $Path_A(M,N)$; $F[P]$ = $Path_A(M,P)$

We have a line $Path(M,N) \to Path(M,P)$
induced by ~~fq~~ a line $N \to P$.
"Ought to be" that we can transport
~~p~~ $\in Path_A(M,N)\,[\Gamma]$ along $Path_A(M, q@x)\,[\Gamma, x]$
to obtain ~~$p \cdot q$~~ $\in Path_A(M,P)\,[\Gamma]$.

Claim: In order for transitivity to hold,
there ought to be coercion.

(13)

Coerce from 0 to 1.

$$\left(\mathrm{coe}^{0\rightsquigarrow 1}_{x.\mathrm{Path}_A(M, q@x)}\right) \in \mathrm{Path}_A(M, q@0) \to \mathrm{Path}_A(M, q@1)$$

If you have an x-line, it ought to go from the left endpt to the right endpt

Coercion along a type line

Intuition: the line will give you everything you need to do the coercion.

Want a generic notion of coercion

$$\mathrm{coe}^{r\to r'}_{x.A} \in A\langle r/x\rangle \longrightarrow A\langle r'/x\rangle$$

(this will end up being an equivalence)

given A type $[\mathcal{I}, x]$
r, r' dim$[\mathcal{I}]$

s.th. same start & end dim.

$$\mathrm{coe}^{r\to r}_{x.A}(M) \doteq M \in A\langle r/x\rangle$$

This is critical

How could this work? coe looks at A to know what to do, and A tells you how you coerce. coe unpacks the equiv given by A, applies it to the left end & gets the right end.

Cases of coercion

Easiest $\mathrm{coe}^{0\rightsquigarrow 0}_{x.A} = id_{A\langle 0/x\rangle}$

$\mathrm{coe}^{1\rightsquigarrow 1}_{x.A} = id_{A\langle 1/x\rangle}$

Easy $\mathrm{coe}^{0\rightsquigarrow 1}_{x.A} : A\langle 0/x\rangle \longrightarrow A\langle 1/x\rangle$ depends on A!

$\mathrm{coe}^{1\to 0}_{x.A} : A\langle 1/x\rangle \longrightarrow A\langle 0/x\rangle$ —— " ——

free/ambient dimension

Harder: $coe^{0\to x}_{x.A} \in A\langle 0/x\rangle \longrightarrow A[\mathbb{I}, x]$

(NB: $x \neq \boldsymbol{x}$, x is bound below & free above)

This is an x-line!

$(coe^{0\to x}_{x.A})\langle 0/x\rangle = coe^{0\to 0}_{x.A} = id$

$(coe^{0\to x}_{x.A})\langle 1/x\rangle = coe^{0\to 1}_{x.A}$ is what it is.

$(coe^{0\to 0}_{x.A})(M_0 \in A\langle 0/x\rangle) = M_0 \in A\langle 0/x\rangle$

$\Big| \; coe^{0\to x}_{x.A}(M_0)$ is the x-line connecting M_0 to what it coerces to. It's a heterogeneous line: the lines are classified by lines

$coe^{0\to 1}_{x.A}(M_0) \in A\langle 1/x\rangle$

$$\begin{array}{ccc} M_0 & \xrightarrow{\quad coe^{0\to x} \quad} & coe^{0\to 1}(M_0) \\ \in & & \in \\ A_0 & \xrightarrow{\quad A \quad} & A_1 \end{array}$$

Next time: $coe^{x\to y}_{x.A}$ ends up being a square.

2018-04-12

Last time

- Cubical "pre-type" theory
  Types & elements are structured as cartesian cubes of arbitrary dimension

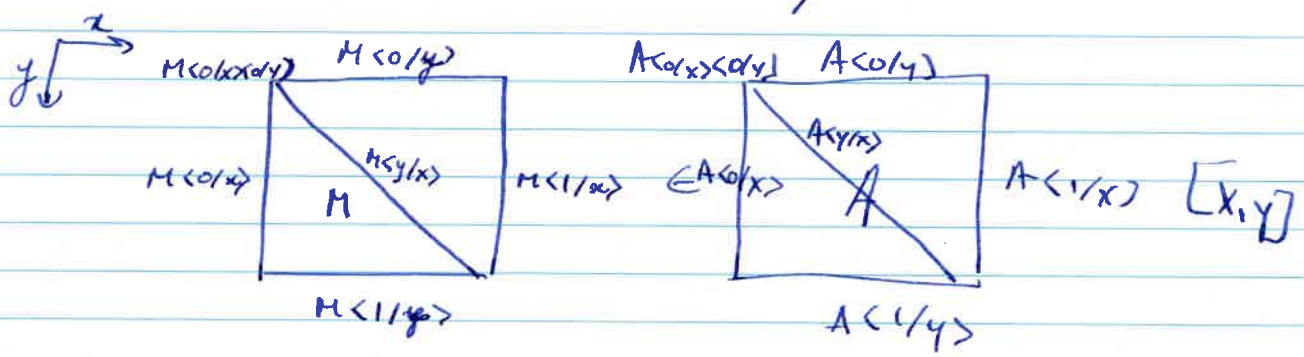
$$A \doteq A' \text{ type } [\Gamma]$$
$$M \doteq M' \text{ type } [\Gamma]$$

NB: hyp judgments work naturally (uniformly) wrt dimension

$$a : A \gg N \in B \; [\Psi]$$

if $\forall \psi : \Gamma' \to \Gamma$ if $M \doteq M' \in A$ then $N[M/x] \doteq N[M'/x] \in [M/x]B$

- Mental picture to keep in mind (the Swedes draw it upside down)



Type lines are what matter!
(Element lines are there to induce type lines)

$$a : A \gg F \text{ type } [\Gamma, x]$$

sends x-lines in A to type lines:
if $M_x \in A \; [\Gamma, x]$, $F[M_x/a]$ type $[\Gamma, x]$

An important "other" type line is given by an equivalence (iso up to path): approx: if "$A \simeq B$ induces $ua_x(E)$ type $[\Gamma, x]$ s.t. $ua_0(E) = A$ & $ua_1(E) = B$"

To be a type (as opposed to a pre-type) requires additional conditions called the Kan conditions:

1. Coercion along a type line

$$\mathrm{coe}^{r \rightsquigarrow r'}_{x.A} : A\langle r/x\rangle \longrightarrow A\langle r'/x\rangle$$

Idea: think of A as induced by F

coe represents the transport in HoTT & does so in 2 stages:

1. M induces a type line
2. The coercion activates (runs it as a program) the type line

$$\mathrm{Coe}^{0 \rightsquigarrow 1}_{x.A} : A\langle 0/x\rangle \longrightarrow A\langle 1/x\rangle$$

$$\approx$$

$$\mathrm{tr}[a.F](P)$$

We examined various coe's.
There's a funny, subtle interplay between type oblivious computation & type awareness.

$\mathrm{coe}^{0 \rightsquigarrow 0}_{x.A}(M) \mapsto M$
$\mathrm{coe}^{1 \rightsquigarrow 1}_{x.A}(M) \mapsto M$
} type-indep

$\mathrm{coe}^{0 \rightsquigarrow 1}_{x.A}(M)$, $\mathrm{coe}^{1 \rightsquigarrow 0}_{x.A}(M)$ } depend on A!

$\mathrm{coe}^{0 \rightsquigarrow y}_{x.A}(M \in A\langle 0/x\rangle) \in A\langle y/x\rangle\,[\mathbb{I}, y]$

these are both y-lines

Remark, they could already have varied in y.

$$\mathrm{coe}^{0 \rightsquigarrow 0}_{x.A\langle 0/y\rangle}\left(M\langle 0/y\rangle \in A\langle 0/y\rangle\langle 0/x\rangle\right) \in A\langle y/x\rangle\langle 0/y\rangle$$
$$A\langle 0,0/x,y\rangle$$
$$A\langle x/y\rangle\langle 0/x\rangle$$
$$\doteq M\langle 0/y\rangle$$

$$\mathrm{coe}^{0 \rightsquigarrow 1}_{x.A\langle 1/y\rangle}\left(M\langle 1/y\rangle \in A\langle 0/x\rangle\langle 1/y\rangle\right) \in A\langle y/x\rangle\langle 1/y\rangle$$
$$= A\langle 1/y\rangle\langle 1/x\rangle$$

↳ deps on $A\langle 1/y\rangle$. Could be computation

due to taking a face. This is why you need coherence: taking faces commutes w/ computation

$\mathsf{coe}^{y \rightsquigarrow 0}_{x.A}$ — symmetric to previous

$\mathsf{coe}^{y \rightsquigarrow y}_{x.A}(M) \longmapsto M$. This should strike you as odd because we're mixing formal & semantic variables. In PL, you can never run when you have variables lying around — they're gone by that point. But here we're talking about equality between variables and it complicates things b/c you need to worry about coherence.

Suppose $y \neq y'$. Then the only possible reductions of $(\mathsf{coe}^{y \rightsquigarrow y'}_{x.A}(M))$ (this is a spoke) are determined by A $\longmapsto^* M'$ (gillion steps)

Now suppose you take

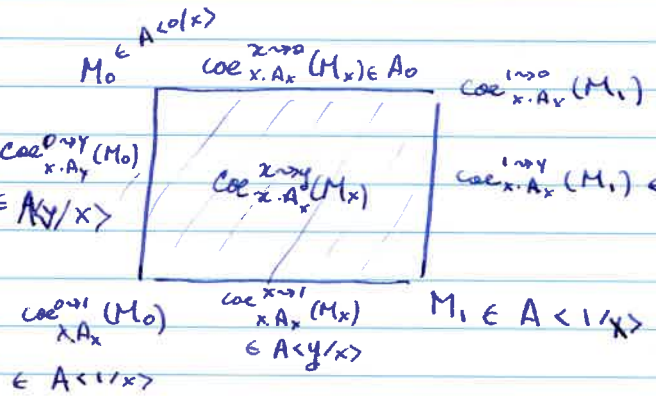
$$\mathsf{coe}^{y \rightsquigarrow y'}_{x.A}(M)\langle y/y' \rangle = \mathsf{coe}^{y \rightsquigarrow y}_{x.A}(M) \longmapsto M$$
$$= \mathsf{coe}^{y \rightsquigarrow y}_{x.A}(M\langle y/y' \rangle) \longmapsto M\langle y/y' \rangle \longmapsto^*$$

and you need to make sure you get the same thing.

Given $M \in A[\mathbb{I}, x]$,

$$M_0 \xrightarrow{M} M_1 \quad \in \quad A_0 \xrightarrow{A} A_1\,[\mathbb{I}, x]$$

The master coe Diagram



Type-specific coercion $\mathrm{coe}^{r \rightsquigarrow r'}_{x.A}(M)$ $(r \neq r')$

Bool is closed, so it doesn't depend on the var.
Let's figure out what this should be:

$$\mathrm{coe}^{r \rightsquigarrow r'}_{-.\,\mathrm{Bool}}(M \in \mathrm{Bool}) \longmapsto M$$

(Bool / Nat / $\mathbb{C}$)

We're in the strict booleans, that have only refl all the way up

No variation $\Rightarrow$ nothing to do.
Now ~~suppose~~ assume we have variation

$$\mathrm{coe}^{r \rightsquigarrow r'}_{x.A \to B}(M : A\langle r/x\rangle \to B\langle r/x\rangle) \in A\langle r'/x\rangle \to B\langle r'/x\rangle \quad [\Psi]$$

$$= \lambda a \in A\langle r'/x\rangle.\ \mathrm{coe}^{r \rightsquigarrow r'}_{x.B}\big(M(\underbrace{\mathrm{coe}^{r' \rightsquigarrow r}_{x.A}(a)}_{\in A\langle r/x\rangle})\big)$$

$\in B\langle r/x\rangle$

$\in B\langle r'/x\rangle$

What makes this correct? It requires a story, which we'll get into later.

$$\mathrm{coe}^{r \rightsquigarrow r'}_{x.(a:A \to B_a)}(M \in a : A\langle r/x\rangle \to B_a\langle r/x\rangle) \in a' : A\langle r'/x\rangle \to B_{a'}\langle r'/x\rangle$$

$\lambda a \in A\langle r'/x\rangle.$

$\mathrm{coe}^{r \rightsquigarrow r'}$? $\big(M(\mathrm{coe}^{r' \rightsquigarrow r}_{x.A}(a))\big)$

$B_a\langle r/x\rangle[\mathrm{coe}^{r' \rightsquigarrow r}_{x.A}(a)/a]$

$B_a\langle r'/x\rangle[a/a]$

We're kind of stuck. We want

$B_a\langle r'/x\rangle[a/a]$ ← but we've changed what we're substituting in for a in B...

Is there any way to get

$$B\langle r/x\rangle[\mathrm{coe}^{r' \rightsquigarrow r}_{x.A}(a)/c] \xrightarrow{?} B\langle r/x\rangle[a/a]$$

Build a line & rely on the action of B on lines!

$$B\langle r/x\rangle[\underbrace{coe^{r'\to x}_{x.A}(a)/a}_{x\text{-line}}] \Big\}\ x\text{-line}$$

Then you get the answer:

$\lambda a \in A\langle r'/x\rangle.$

$$\underbrace{coe^{r\to r'}_{x.B_x[coe^{r'\to x}_{x.A}(a)/a]}}_{x\text{-line}}\Big(\underbrace{M\,(coe^{r'\to r}_{x.A}(a))}_{B\langle r/x\rangle[coe^{r'\to r}_{x.A}(a)/a]}\Big)$$

$$B\langle r'/x\rangle[a/a]$$

Exercises:

1. $coe^{r\to r'}_{x.(A\times B)}\,(M \in A\langle r/x\rangle \times B\langle r/x\rangle) \in A\langle r'/x\rangle \times B\langle r'/x\rangle$

2. $coe^{r\to r'}_{x.(a:A\times B)}\,(M \in a:A\langle r/x\rangle \times B_a\langle r/x\rangle) \in a:A\langle r'/x\rangle \times B_a\langle r'/x\rangle.$

Just carefully follow your nose.

Consider now

$$coe^{r\to r'}_{x.Path_{y.A}(P_x,Q_x)}\,(R \in Path_{y.A\langle r/x\rangle}(\underbrace{P\langle r/x\rangle}_{A\langle 0/y\rangle\langle r/x\rangle}, \underbrace{Q\langle r/x\rangle}_{A\langle 1/y\rangle\langle r/x\rangle}))$$

$$\in Path_{y.A\langle r'/x\rangle}(\underbrace{P\langle r'/x\rangle}_{A\langle 0/y\rangle\langle r'/x\rangle}, \underbrace{Q\langle r'/x\rangle}_{A\langle 1/y\rangle\langle r'/x\rangle})$$

Recall, these are quoted paths of the form $\langle y.p\rangle$

This is non-trivial & forces us to consider another condition on paths called Kan composition

Then we will get that a

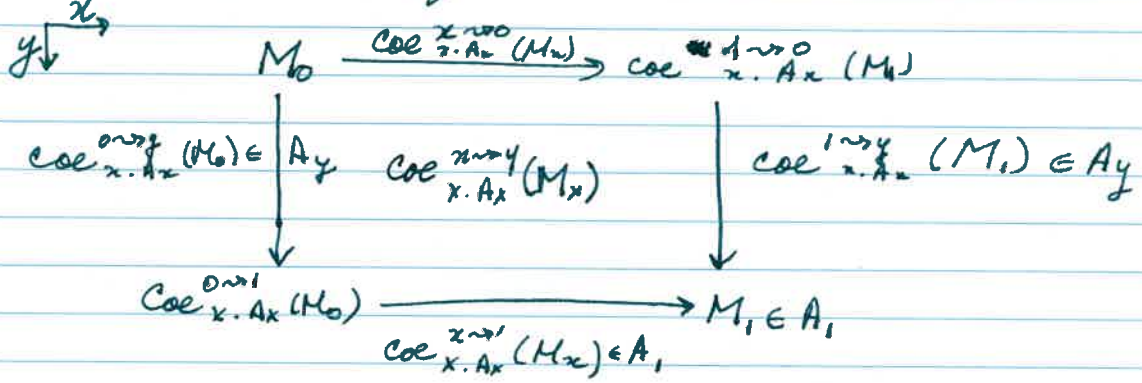
Type = pre-type
+ Kan coercion
+ Kan composition (enough lines)

2018-04-24

Given $M \in A[\Psi, x]$

$$M_0 \xrightarrow{M} M_1 \in A_0 \xrightarrow{A} A_1$$

we have the square



(In $\mathrm{coe}^{x \rightsquigarrow 0}_{x.A_x}(M_x)$, the x in "$x \rightsquigarrow 0$" & "M_x" are free, and the x in $x.A_x$ is bound)

For all types A, $\mathrm{coe}^{r \rightsquigarrow r}_{x.A}(M) \mapsto M$. The other cases depend on A. → Realizing being able to $x \rightsquigarrow x$ is the same variable is important

$\mathrm{coe}^{r \rightsquigarrow r'}_{x.\mathrm{Bool}}(M) \mapsto M$

$\mathrm{coe}^{r \rightsquigarrow r'}_{x.\mathbb{C}}(M) \mapsto M$

$\mathrm{coe}^{r \rightsquigarrow r'}_{x.\mathbb{N}ar}(M) \mapsto M$

$\mathrm{coe}^{r \rightsquigarrow r'}_{x.A \to B}(M) \in A\langle r'/x\rangle \to B\langle r'/x\rangle$

$\triangleq \lambda a \in A\langle r'/x\rangle.\ \mathrm{coe}^{r \rightsquigarrow r'}_{x.B}(M(\mathrm{coe}^{r' \rightsquigarrow r}_{x.A}(a)))$ ← contra, cov →

$\mathrm{coe}^{r \rightsquigarrow r'}_{x.(a:A) \to B(a)}(M \in a:A\langle r/x\rangle \to B(a)\langle r/x\rangle)$

$\triangleq \lambda a \in A\langle r'/x\rangle.\ \mathrm{coe}^{r \rightsquigarrow r'}_{x.B[\mathrm{coe}^{r' \rightsquigarrow x}_{x.A}(a)/a]}(M(\underbrace{\mathrm{coe}^{r' \rightsquigarrow r}_{x.A}(a)}_{\in A\langle r/x\rangle}))$

$\in B[\mathrm{coe}^{r \rightsquigarrow r}_{x.A}(a)/a]\langle r/x\rangle$

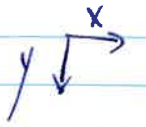
$= B\langle r/x\rangle[\mathrm{coe}\ldots/a]$

$\mathrm{coe}^{r' \rightsquigarrow x}_{x.A} \in A\langle r'/x\rangle \to A\langle x/x\rangle$

$\mathrm{coe}^{r' \rightsquigarrow r}_{x.A}(a) \quad \mathrm{coe}^{r' \rightsquigarrow x}_{x.A}(a) \quad \mathrm{coe}^{r \rightsquigarrow r'}_{x.A}(a) = a$

$B[\ \ /a] \quad B[\ \ /a] \quad B\langle r'/x\rangle[\ \ /a] =$

$= B\langle r/x\rangle[a]$



$\mathrm{coe}^{r\to r'}_{y.\,\mathrm{Path}_{x.A}(P_0,P_1)} \in \mathrm{Path}_{x.A\langle r/y\rangle}(P_0\langle r/y\rangle, P_1\langle r/y\rangle) \to \mathrm{Path}^{(P_0\langle r'/y\rangle, P_1\langle r'/y\rangle)}_{x.A\langle r'/y\rangle}$ [Ψ]

where $P_0 \in A\langle 0/x\rangle$, $P_1 \in A\langle 1/x\rangle$ [Ψ, y]

A type [Ψ, x, y]

Given $M \in \mathrm{Path}_{x.A\langle r/y\rangle}(P_0\langle r/y\rangle, P_1\langle r/y\rangle)$ [Ψ],

we know $M@0 \doteq P_0\langle r/y\rangle$ —$M@x$— $M@1 \doteq P_1\langle r/y\rangle \in A\langle r/y\rangle$ [Ψ, x]

Want

$P_0\langle r/y\rangle$ —?— $P_1\langle r/y\rangle$ $\in A\langle r'/y\rangle$ [Ψ, x]

$P_0\langle r'/y\rangle \in A\langle 0/x\rangle\langle r'/y\rangle$ —?— $P_1\langle r'/y\rangle \in A\langle 1/x\rangle\langle r'/y\rangle$

Then $\langle x.\ ?\rangle \in \mathrm{Path}_{x.A\langle r'/y\rangle}(P_0\langle r'/y\rangle, P_1\langle r'/y\rangle)$ [Ψ],

which is what we want.

What if we took the line

(*) $\mathrm{coe}^{r\to r'}_{y.A}(M@x) \in A\langle r'/y\rangle$ [Ψ, x]

$\mathrm{coe}^{r\to r'}_{y.A\langle 0/x\rangle}(M@0)$ —— $\mathrm{coe}^{r\to r'}_{y.A\langle 1/x\rangle}(M@1)$?

$M@0 \doteq P_0\langle r/y\rangle$; $\mathrm{coe}^{r\to r'}_{y.A\langle 0/x\rangle}(M@0) \in A\langle 0/x\rangle\langle r'/y\rangle$ [Ψ]

$M@1 \doteq P_1\langle r/y\rangle$; $\mathrm{coe}^{r\to r'}_{y.A\langle 1/x\rangle}(M@1) \in A\langle 1/x\rangle\langle r'/y\rangle$ [Ψ]

We know that $P_0 \in A\langle 0/x\rangle$ [Ψ, y], so $P_0\langle r/y\rangle \in A\langle 0/x\rangle\langle r/y\rangle$ [Ψ]

$P_1 \in A\langle 1/x\rangle$ [Ψ, y], so $P_1\langle r/y\rangle \in A\langle 1/x\rangle\langle r/y\rangle$ [Ψ].

The endpoints of (*) are in the right type, but the endpoints aren't the ones we want ($P_i\langle r'/y\rangle$ vs. $\mathrm{coe}^{r\to r'}_{y.A\langle i/x\rangle}(M@i)$).

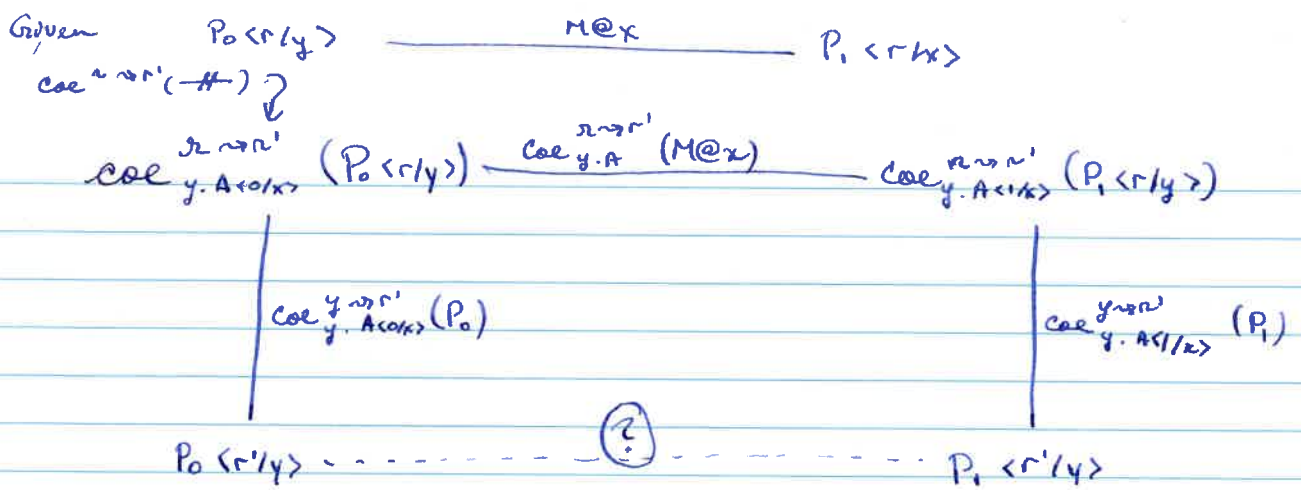
Note: 1) $\mathrm{coe}^{y\to r'}_{y.A\langle 0/x\rangle}(P_0) \in A\langle 0/x\rangle\langle r'/y\rangle$ [Ψ, y]

2) $\mathrm{coe}^{y\to r'}_{y.A\langle 1/x\rangle}(P_1) \in A\langle 1/x\rangle\langle r'/y\rangle$ [Ψ, y].

When $y = r$, we get

1) $\mathrm{coe}^{r\to r'}_{y.A\langle 0/x\rangle}(P_0\langle r/y\rangle)$ when $y = r'$ $\mathrm{coe}^{r'\to r'}_{y.A\langle 0/x\rangle}(P_0\langle r'/y\rangle) \mapsto P_0\langle r'/y\rangle$

2) $\mathrm{coe}^{r\to r'}_{y.A\langle 1/x\rangle}(P_1\langle r/y\rangle)$ $\mathrm{coe}^{r'\to r'}_{y.A\langle 1/x\rangle}(P_1\langle r'/y\rangle) \mapsto P_1\langle r'/y\rangle$



If lines are data witnessing the interchangeability of endpts, it stands to reason that we should be able to interchange the $P_i\langle r'/y\rangle$. To get this, we require our types to have/satisfy Kan composition, which we introduce now:

$$? = \mathrm{hcom}^{r\rightsquigarrow r'}_{A\langle r'/y\rangle}\left(\underbrace{\mathrm{coe}^{r\rightsquigarrow r'}_{y.A}(M@x)}_{\text{"cap" or "top"}}; \underbrace{\begin{cases} x=0 \hookrightarrow y.\mathrm{coe}^{y\rightsquigarrow r'}_{y.A\langle 0/x\rangle}(P_0) \\ x=1 \hookrightarrow y.\mathrm{coe}^{y\rightsquigarrow r'}_{y.A\langle 1/x\rangle}(P_1) \end{cases}}_{\text{the "system", which describe what to glue at the endpts}} \right)$$

Cleaner notation: "$\mathrm{com}^{r\rightsquigarrow r'}_{y.A}(M@x; \{x=0 \hookrightarrow y.P_0 \;\; x=1 \hookrightarrow y.P_1\})$"

and then the image of M becomes the quotation of this Composition.

Coe = what do you do w/ the lines you have
hcom = You have enough lines.

2018-04-26

Topic: Kan conditions on pre-types sufficient to be types

Pretypes — cubical structure
- ramifications into dim
- higher dims are supposed to represent identifications (evidence for interchangeability)

"Why?" 1. coercion along a type identification
$\mathrm{coe}^{r \rightsquigarrow r'}_{x.A}(M)_{\in A\langle r/x\rangle} \in A\langle r'/x\rangle$

"What?" 2. composition of identifications
$\mathrm{hcom}^{r \rightsquigarrow r'}_{A}(M; \xi_i \hookrightarrow y.N_i)$

A motivation for composition: derive coe in a Path type.

$\mathrm{coe}^{r \rightsquigarrow r'}_{y.A\langle 0/x\rangle}(P_0\langle r/y\rangle)$ $\quad \mathrm{coe}^{r \rightsquigarrow r'}_{y.A}(M@x) \quad$ $\mathrm{coe}^{r \rightsquigarrow r'}_{y.A\langle 1/x\rangle}(P_1\langle r/y\rangle)$

$\mathrm{coe}^{y \rightsquigarrow r'}_{y.A\langle 0/x\rangle}(P_0)$ | | |

$P_0\langle r'/y\rangle$ - - - - - (?) - - - - - $P_1\langle r'/y\rangle$

$\mathrm{coe}^{r \rightsquigarrow r'}_{y.\mathrm{Path}_{x}(P_0,P_1)}(M) = (?)$

The identification we want is notated thus:

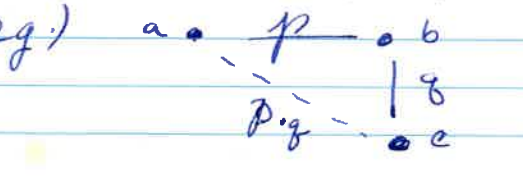
$\mathrm{hcom}^{r \rightsquigarrow r'}_{A}\Big(\overbrace{\mathrm{coe}^{r \rightsquigarrow r'}_{y.A}(M@x)}^{\text{"cap"}}; \overbrace{\begin{cases} x=0 \hookrightarrow y.\mathrm{coe}^{y \rightsquigarrow r'}_{y.A\langle 0/x\rangle}(P_0) \\ x=1 \hookrightarrow y.\mathrm{coe}^{y \rightsquigarrow r'}_{y.A\langle 1/x\rangle}(P_1) \end{cases}}^{\text{"tubes"}}\Big)$

↑ homogeneous (We have implicit adjacency constraints that when $x=0$ & $y=k$, then the subtypes are the same, & similarly when $x=1$ & $y=k$, that is, the lines meet up in the corners in the above diagram)

aka $\mathrm{com}^{r \rightsquigarrow r'}_{x.A}\Big(M@x; \begin{cases} x=0 \hookrightarrow y.P_0 \\ x=1 \hookrightarrow y.P_1 \end{cases}\Big)$ ← Derived form

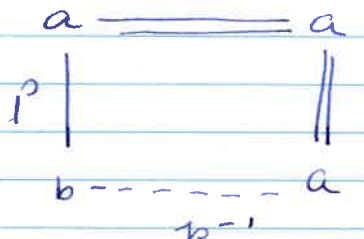
↳ see (137)

We claim this is the general form/case.

eg.) 

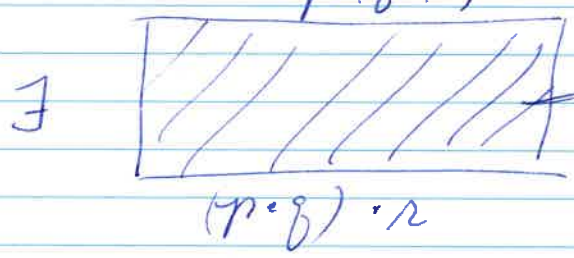
Transitivity / concat

eg)



Symmetry, aka reversal, inversion

(Can derive higher identifications witnessing groupoid laws



line between lines)

The validity of such an identification is due to a higher ident, which holds due to a higher identification, and it's "lies all the way up" and nobody can ever catch you out.

Two ways you can justify this:

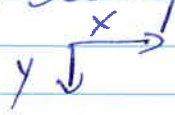
1. Interpret in terms of some more complicated theory.
2. Accept it via a computational outlook (The only way we finite beings can speak of the infinite is via computation.)

Kan conditions (to be a type)

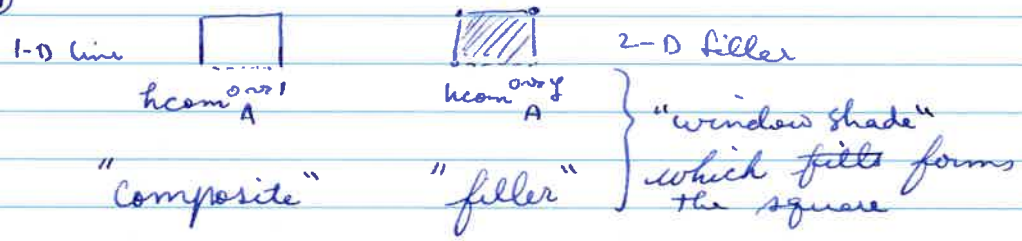
A type $[\mathbb{F}]$ means

1) $A \Vdash A_0$ and A_0 is a canonical type in $\mathbb{F}$
2) coercion structure
   if $\mathbb{F} = \mathbb{F}', x$, then for all r, r'
   $coe^{r \to r'}_{x.A} \in A\langle r/x\rangle \to A\langle r'/x\rangle$
   when $coe^{r \to r}_{x.A} = id_{A\langle r/x\rangle}$ $[\mathbb{F}]$

3) composition structure



The following exist



$hcom^{0 \rightsquigarrow 1}_{A}$ $hcom^{0 \rightsquigarrow y}_{A}$

$hcom^{y \rightsquigarrow 0}$ $hcom^{y \rightsquigarrow y'}$?

Special case

Given $M \in A[\mathcal{I}, x]$ "cap"

$N_0 \in A[\mathcal{I}, x, y \mid x=0]$ "tube"

$N_1 \in A[\mathcal{I}, x, y \mid x=1]$ "tube"

s.t. $N_0\langle r/y\rangle \doteq M \in A[\mathcal{I}, x \mid x=0]$

~~we obtain~~ $N_1\langle r/y\rangle \doteq M \in A[\mathcal{I}, x \mid x=1]$

we obtain

$$hcom^{r \rightsquigarrow r'}_{A}(M; \{x=0 \hookrightarrow y.N_0,\ x=1 \hookrightarrow y.N_1\}) \in A[\mathcal{I}, x]$$

s.t.

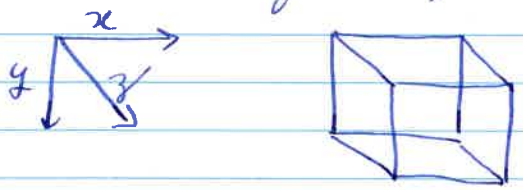
if $r=r'$, then $hcom(=) \doteq M \in A[\mathcal{I}, x)$

if $r \neq r'$, then $hcom(\neq) \doteq N_0\langle r'/y\rangle \in A[\mathcal{I}, x \mid x=0]$

$hcom(\neq) \doteq N_1\langle r'/y\rangle \in A[\mathcal{I}, x \mid x=1]$

This covers the picture on (134)

What happens when you go up one dimension and the cap is a square, the composite is a square, and the filler is a cube?



z { back face "cap"; front face "comp"

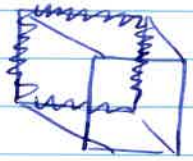
xy

Top, bottom, L, R faces = tube

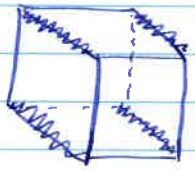
Entire cube = filler

You have 2 sets of adjacency constraints:

1) the tube faces must be adj to the cap

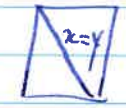


2) the tube faces must be adjacent to each other



A general naturality condition tells us that we should be able to do things incrementally or all at once & get the same thing: eg. Start w Back, Top, Bottom then fill in sides then get front vs. Start out w the 7 faces & then do the front.

What about diagonals?



They are a way of expressing coherence!!! This will be the key to making univalence work.

General Case — This is a theorem about the semantics.

cf A type $[\mathcal{I}]$
$M \in A\ [\mathcal{I}]$
$\overrightarrow{r_i = r_i'}$ valid — p. (137)

(cap adj) $(\forall i)\ N_i\langle r/y\rangle \doteq M \in A\ [\mathcal{I} \mid r_i = r_i']$
(tube adj) $(\forall i,j)\ N_i \doteq N_j \in A\ [\mathcal{I} \mid r_i = r_i', r_j = r_j']$

then $\mathrm{hcom}_A^{r\,r'}(M;\ \overrightarrow{r_i = r_i' \hookrightarrow y.N_i}) \in A\ [\mathcal{I}]$
s.th if $r = r'$, then $\mathrm{hcom}(\cdots) \doteq M \in A\ [\mathcal{I}]$
if $r \neq r'$, then $\mathrm{hcom}(\cdots) \doteq N_i\langle r'/y\rangle \in A\ [\mathcal{I} \mid r_i = r_i']$

$\overrightarrow{r_i = r_i'}$ valid :

either

1) r_i is r_i' $(\exists i)$
   $0 = 0$, $1 = 1$, $x_i = x_i$

or

2) r_i is r_j, r_i' is 0 r_j' is 1 $(\exists i,j)$

   lhs same

$x = 0$ ⇓ x_i $\quad$ $x = 1$ ⇓ x_j

What are the computation rules?

$hcom_A^{r \rightsquigarrow r} (M; \_\_) \longmapsto M$

$hcom_A^{r \rightsquigarrow r'} (M; \overrightarrow{r_i = r_i \hookrightarrow y.N_i}, \_\_) \longmapsto N_i \langle r'/y \rangle$ — left-most, b/c we want a deterministic op. sem.

$hcom_A^{r \rightsquigarrow r'} (M; \overrightarrow{\xi_i \hookrightarrow y.N_i}) \longmapsto fcom^{r \rightsquigarrow r'} (M; \overrightarrow{\xi_i \hookrightarrow y_i.N_i})$ val

"free composition"
"formal composition"

(Q: What is if supposed to do when it encounters one of these formal objects? if thinks there are only two bools.)

2018-05-01

Last time: Kan conditions

(What) 1. Coercions $\mathrm{coe}^{r\rightsquigarrow r'}_{x.A} : A\langle r/x\rangle \longrightarrow A\langle r'/x\rangle$
s.th. $\mathrm{coe}^{r\rightsquigarrow r}_{x.A}(M) \mapsto M$

(Which) 2. Composition $\mathrm{hcom}^{r\rightsquigarrow r'}_{A}(M; \overrightarrow{\xi_i \longrightarrow y.N_i})$

[ξ_i are are dim equations s.th. one is true or there is a pair $r=0$, $r=1$.

$\mathrm{hcom} \in A[\mathcal{F}]$ when
$M \in A[\mathcal{F}]$
$N_i \in A[\mathcal{F}, y \,|\, \xi_i]$

adjacency
$N_\varepsilon\langle r/y\rangle \doteq M\langle \varepsilon/x\rangle \in A$
$N_i \doteq N_j \in A[\Psi \,|\, \xi_i\, \xi_j]$

i.H.
$\mathrm{hcom}^{r\rightsquigarrow r'}_{A}(M; -) \mapsto M$
$\mathrm{hcom}^{r\rightsquigarrow r'}_{A}(M; \cdots \underbrace{\xi_i}_{\text{left/mor true}} \longrightarrow y.N_i \cdots) \mapsto N_i\langle r'/y\rangle$

(Visually, it helps to think of the cube and the 0, 1 cases)

Defining hcom_A for various A

1) strict booleans ("built-in"/"by hand")

Bool — observables for canonicity: closed terms should evaluate to true or false. Implies termination.

Bool val — $\mathbb{B}\mathrm{ool} \doteq \mathbb{B}\mathrm{ool}$ type $[\mathcal{F}]$
tt, ff val — $\mathrm{tt} \doteq \mathrm{tt} \in \mathbb{B}\mathrm{ool}\,[\mathcal{F}]$
$\mathrm{ff} \doteq \mathrm{ff} \in \mathbb{B}\mathrm{ool}\,[\mathcal{F}]$

$\mathrm{if}_{x.A}(M; M_0, M_1) \mapsto \mathrm{if}_{x.A}(M'; M_0; M_1)$ if $M \mapsto M'$

$\mathrm{if}(\mathrm{tt}; M_0; M_1) \mapsto M_0$
$\mathrm{if}(\mathrm{ff}; M_0; M_1) \mapsto M_1$

We claim this is a Kan type, so we must

give compositors.

$\mathrm{coe}^{r \rightsquigarrow r'}_{\cdot : \mathrm{Bool}}(M) \mapsto M$

$\mathrm{hcom}^{r \rightsquigarrow r'}_{\mathrm{Bool}}(M, \xi_i \hookrightarrow y.N_i) \mapsto M$

No lines! (Other than degeneracies)

Check that the required conditions hold to be a valid Kan structure!
(This is so degenerate in every way that this is almost automatically Kan)

2) wBool weak booleans ("ind. defined")

"the ~~least~~ free Kan type generated by true & false generators: $\underbrace{\mathrm{tt}, \mathrm{ff}}_{\mathrm{val}_I} \in \mathrm{wBool}\ [I]$

For technical reasons, last time we had hcom $\mapsto$ from val. We'll just do, for today, the following:

$$\mathrm{hcom}^{r \rightsquigarrow r'}_{\mathrm{wBool}}(M; \xi_i \hookrightarrow y.N_i)\ \mathrm{val}_I$$

a | tt / tt | tt $\leftarrow$ H | ff / ff | ff H } We are oblivious of the fact that adjacency requirements force any squares to be uninteresting instance of ttttt or ffffff. So we just throw in the hcom as a formal object & call it a day.

i.e. There are infinitely many booleans!

$\mathrm{if}_{a.A}(M; M_0; M_1) \in A[M/a]\ [I]$
"property of M"
["Motive"

provided $\{ M_0 \in A[\mathrm{tt}/a] \quad M_1 \in A[\mathrm{ff}/a]\ [I]$
$a : \mathrm{Bool} \gg A\ \mathrm{Type}$

But what do you do for

$$if_{a.A}\Big(\underbrace{hcom^{r\rightsquigarrow r'}_{wBool}(M;\ \xi_i \hookrightarrow y.N_i)}_{\in wBool\,[\mathbb{I},x]};\ M_0;M_1\Big) \longrightarrow\ ?$$

The hcom is an x-line. This feels a lot like the embarassing situation we were in where we had inhabitants of an identity type that ~~weren't~~ weren't refl.

What to do?

$$if_{a.A}(-;M_0;M_1) = F(-)$$

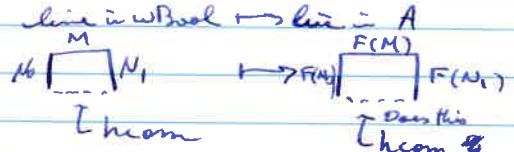
Want $\quad b : Bool \gg F(b) \in A[b/a]\ [\mathbb{I}]$

What do we know about these mappings?
We have a presheaf semantics!
The mapping → sends points to points
→ sends lines to lines
↳ preserves identifications!

$if_{a.A}(hcom(--);\ M_0;M_1) \longrightarrow\ ?$

↑
Want to preserve the hcom line.
What does this mean?
"$F \in wBool \to A$"

line in wBool ↦ line in A



This is not quite right, see (143)

→ takes advantage of the motive

$$F\left(hcom^{r\rightsquigarrow r'}_{wBool}(M;\ \overrightarrow{\xi_i \hookrightarrow y.N_i})\right) \mapsto$$

$$\mapsto \cancel{\text{hcom}}\ hcom^{r\rightsquigarrow r'}_{A[\frac{?}{a}]}\ (F(M);\ \overrightarrow{\xi_i \hookrightarrow y.F(N_i)}) \in$$

$\in A[M/a]$ ← different types → $\in A[N_i/a]$

(X) ~~Doesn't type check~~

$$\in A[hcom^{r\rightsquigarrow r'}_{wBool}(M;\ \overrightarrow{\xi_i \hookrightarrow y.N_i})/a]$$

~~Doesn't~~

Can assume A is Kan, because our whole setup assumes we live in a world where all types are Kan
⇒ A has composition!
Take the line in wBool and ask A to make a line out of it. (142)
Need to know the motive!

What should ⊛ be?

Let's look at the z line

⊕ $\mathrm{hcom}^{r \rightsquigarrow y}_{\mathbb{W}\mathrm{Bool}}(M;\ \xi_i \hookrightarrow y.N_i)$

z line @ $r \doteq M$

z line @ $r'(\neq r) \doteq \mathrm{hcom}^{r \rightsquigarrow r'}_{\mathbb{W}\mathrm{Bool}}$ (——)

So we have a z line

M —————— $\mathrm{hcom}^{r \rightsquigarrow r'}_{\mathbb{W}\mathrm{Bool}}$ (——)

$A[\mathrm{hcom}^{r \rightsquigarrow r}(-)]$ —— $A[\mathrm{hcom}^{r \rightsquigarrow y}(-)]$ —— $A(\mathrm{hcom}^{r \rightsquigarrow r'}(-)]$

Need to show $F(M) \in A[\mathrm{hcom}^{r \rightsquigarrow y}(-)]\langle r'/z\rangle \overset{\langle r'/z\rangle}{=} A[M]$

$F(N_i) \in A[\text{————}]\langle r'/z\rangle = A(N_i)$

Because the type varies in z, we can't use the hcom in ⊕ for ⊛.

Rather, we must generalize to a com:

Heterogeneous composition

$\mathrm{com}^{r \rightsquigarrow r'}_{\underbrace{z.A}_{\text{line}}}(M;\ \xi_i \hookrightarrow y.N_i)$

$\triangleq \mathrm{hcom}^{r \rightsquigarrow r'}_{A\langle r'/z\rangle}(\underbrace{\mathrm{coe}^{r \rightsquigarrow r'}_{z.A}(M)}_{\in A\langle r'/z\rangle};\ \xi_i \hookrightarrow y.\underbrace{\mathrm{coe}^{r \rightsquigarrow y}_{z.A}(N_i)}_{A\langle y/z\rangle})$

So we get:

$F(\mathrm{hcom}^{r \rightsquigarrow r'}_{\mathbb{W}\mathrm{Bool}}(M;\ \xi_i \hookrightarrow y.N_i)) \longmapsto$

$\longmapsto \mathrm{com}^{r \rightsquigarrow r'}_{z.A[\oplus/z]}(F(M);\ \xi_i \hookrightarrow y.F(N_i))$

So, in short, we know what to do on $\mathbb{W}$, $\mathbb{H}$, and when we see an hcom, we just

rotate the composite out / shift it off onto A.

Remark: Can't do any of this if you don't know what the motive is!

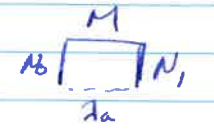
Let's look at more examples.

3) $A \to B$ (no dep)

$\mathrm{hcom}^{r \rightsquigarrow r'}_{A \to B}\ (M^{\in A \to B},\ \xi_i \hookrightarrow y.N_i^{\in A \to \mathcal{U}[\xi_i]})$ "a line of functions is a function of lines"

$\in A \to B$

$\mapsto \lambda a.\ \mathrm{hcom}^{r \rightsquigarrow r'}_{B}\ (M(a)^{\in B},\ \xi_i \hookrightarrow y.N_i(a)^{\in B})$



Adjacency still holds because if the ~~two~~ functs are adj, then they remain adj when applied.

4) $a : A \to B(a)$

$\mathrm{hcom}^{r \rightsquigarrow r'}_{a:A \to B(a)}\ (M^{\in a:A \to B(a)},\ \xi_i \hookrightarrow y.N_i^{\in a:A \to B(a)[\xi_i]})$

$\mapsto \lambda a.\ \mathrm{hcom}^{r \rightsquigarrow r'}_{B(a)}\ (M(a)^{\in B(a)},\ \xi_i \hookrightarrow y.N_i(a)^{\in B(a)})$

5) $a : A \times B(a)$ Need to be careful: dependency trips you up. ~~No dependency, just map the dep.~~

$\mathrm{hcom}^{r \rightsquigarrow r'}_{a:A \times B(a)}\ (M;\ \overrightarrow{\xi_i \hookrightarrow y.N_i})$ $\in a : A \times B(a)$

$\xrightarrow[A \times B]{\text{No dep}} \langle \mathrm{hcom}^{r \rightsquigarrow r'}_{A}\ (\mathrm{fst}(M),\ \overrightarrow{\xi_i \hookrightarrow y.\mathrm{fst}(N_i)}),\ \mathrm{hcom}^{r \rightsquigarrow r'}_{B}\ (\mathrm{snd}(M),\ \overrightarrow{\xi_i \hookrightarrow y.\mathrm{snd}(N_i)}) \rangle$

heterogeneous com!

$\xrightarrow[a:A \times B(a)]{\text{w dep}} \langle \mathrm{hcom}^{r \rightsquigarrow r'}_{A}\ (\mathrm{fst}(M),\ \overrightarrow{\xi_i \hookrightarrow y.\mathrm{fst}(N_i)}),\ \mathrm{com}^{r \rightsquigarrow r'}\ (\mathrm{snd}(M),\ \overrightarrow{\xi_i \hookrightarrow y.\mathrm{snd}(N_i)}) \rangle$

$\mathrm{J.B}[\mathrm{hcom}^{r \rightsquigarrow r'}_{A}(\mathrm{fst}(M),\ \overrightarrow{\xi_i \hookrightarrow y.\mathrm{fst}(N_i)})/a]$

6) Check for yourself: $\mathrm{Path}_{x:A}(P;Q)$

$$\mathrm{hcom}^{r\rightsquigarrow r'}_{\mathrm{Path}_{x:A}(P;Q)}\left(M;\ \overrightarrow{\xi_i \hookrightarrow_y N_i}\right)$$

Hint: Dimension shift: augment constraints $\tilde{\omega}$ $x=0$, $x=1$.