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. Realizability implementation of CTT
2. Guarded higher comp'F type theory
3. Higher type theory / cubical type theory
   a) univalence, inductive types
4. CTT as a type language for 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 FTs:
   CCHL, ABCFHL (cartesian) (CHL)
   (CCHL, weak extension)

6) Nanevski (IMDEA)'s type theory for concurrency, which seems to be computational, no session
   types which are formal/C-H

Motivated.
Formal IT
also structural
prescriptive
deductive

formal logic
You're defining an R.E. set.
Inductively defining
"judgment" (Kreis thinks
this is verifiable empirically)
Γ∪A types defining terms
Γ⊢M:A and types.


Defined by a collection of
rules. Often squiggly
with a notion of equality
("definitional equality")
Γ⊢A=B, Γ⊢M=N:A

Then often capture comp.
where typing, etc., are stable
under definitional quality.
Γ⊢M:A, Γ⊢A=B
Γ⊢M:B.

The methodology:
1. You have a choice to formal
   logic. Γ⊢A prop.
   Γ⊢A type,
   as often by construction.
   But then you have stage
   types like nat and list.
   Elements of a type are programs
   that satisfy or adhere to the
   type's given by the type (intepreted)
   
2. The ought to be constructive
   in two senses (?);
   all the proof terms have
   a computational is
   interpretation and
   carry to programs. Or
   so you would like.
   many fail at this, like HoTT.
(Absence of LEM or DNE or… This is not desired for axiomatization: freedom is the fewer of structure.

The role of proof theories is to explore the structure, not to answer definitive questions of existence. The proof theories of a computational world are inductively defined. The computational world is inductively defined.

- $A \Rightarrow A$ Left, $M$ Right
- $A \Rightarrow A$ is true in extension

3. Decidability. The judgments are computable. These are examples of a computable logic (Curry and Belnap).

- Decidability: $\Gamma \vdash A$ or $\neg A$
- $\Gamma \vdash A$ or $\neg A$
- $\Gamma \vdash A$ or $\neg A$

idea: Type checking is proof checking. (The complexity of the decision problem is horrendous beyond belief — it is considerably.

4. Computational meaning. If any, is given by the fact of the fact what they're really interested in is the interpretation of their formal system into various mathematical structures, à la model theory.

- $\vdash A \Rightarrow A$ in $\text{nat}$
Case Study: Simply Typed A-calculus \( \Gamma \vdash A \rightarrow B \).

**Function types**
- **Base type:** for illustrating terms
  - **Type:** \( \text{Type } A = b \mid A \rightarrow B \) - 2 distinct inhabitants
- **Terms:** \( M = z \mid e \mid 1 \times A.M \mid MN \)
- **Context:** \( \Gamma = - \mid \Gamma; x : A \) distinct variables

**Fr M ; A is the typing judgment**

**Fr M = N ; A is the definitional equality**

In particular, closed terms \( M : A \) (perhaps bare type \( A \))

**Technical stuff:** \( \Gamma(M) = \text{substitution } \)

\[ \begin{align*}
\text{if } \forall \alpha \in A \rightarrow A, \alpha \text{ is a term } x : A \rightarrow A, \\
\Gamma(M) \text{ extends to terms.}
\end{align*} \]

Any notion of computation comes from the outside and is extrinsic.

**Fr M = N ; A includes computation and**

in Reflex Sym Trans Congruence

**For example:**

\[ \begin{align*}
\Gamma, x : A.M : B & \quad \Gamma, x : M : A \\\n\Gamma, x : A.M : B & \quad \Gamma, x : M : B
\end{align*} \]

**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 (terms)

2. Open terms of any type and open reduction. Variables range over open terms.

**Intuition:**

If \( x : A.M : B \), \( x \) is restricted to closed terms, equality much fewer things are going to be equal. i.e.,

\[ \text{x:nat} \vdash M = N : \text{nat} \]

Only for \( M, N \) constant natural numbers.
Our open terms, then you can get
\[ x \cdot \text{nat} + x + x = 2x \cdot \text{nat}. \]

1. Closed computation

We'll define 3 judgments:

\[ \text{val} \hspace{1cm} M \text{val} \Rightarrow M \rightarrow \nu V \text{val} \]
\[ M \rightarrow M' \hspace{1cm} M \rightarrow M' \]
\[ c \text{ val} \hspace{1cm} c \cdot A.M \text{ val} \Rightarrow H.N \rightarrow M' \]

\( (A \cdot c \cdot A.M) N \rightarrow [\nu N/c] M \)

Evaluation contexts:

\( \xi \vdash M \) "spine"

Define \( \xi \vdash M \) to fill the to hole \( \nu M \)

Don't need to worry much because we're dealing only with closed terms.

We can now write

\[ M \rightarrow M' \text{ if } M = \xi \vdash P \]

Felleiier thinks ESC is easier to formalize
But thinks SOS is easier to formalize
and is what he and two used to formalize
Still in twenty...

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

Claim: If \( M : A \) then \( M \cdot \nu (\exists V \cdot \text{val} V) \)
We will then want to show:

**Goal:** Show that all well-typed open terms normalize.

If \( M : \text{M} \) then \( M \) normalizes.

**Proof:** By induction on syntax of \( M \).

**Var:** impossible

**Const:** \( M = c \): The \( A = b \) and \( c = \text{val} \)

**App:** \( M = NP \), \( N = B \rightarrow A \), \( NV \)

**P:** \( B \), \( P \), \( P \), \( P \)

\( \text{val hyp} \)

**T9:** \( NP \), \( NV \). Not obvious.

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

**By:** 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 title! (Tcha dae to Vanil?!)  

Define a property called *recursively terminating* \( HT(M) \) s.t.

1. \( \text{implies \ termination} \)
2. \( \text{induction goes through} \)

\( HT(M) \) iff \( M \) behaves as a structural core.

\( HT_{A \rightarrow B} (M) \) iff \( VN \), \( (HT(N) \text{ implies } HT_B(M)) \)

This is called a "negative" formulation.

There exists a positive formulation.
For fun, imagine we had polymorphism $\text{HA}_b A (M)$?

Claim: If $M : A$ then $\text{HA}_b A (M)$

Proof:

Var: $\lambda x. y : \text{MT}_b A, y : A + x : A$

Body: $A : x : A + y : A, A : A + x : A$

App: $\text{HA}_b A (M)$ : $\text{HA}_b A (M)$

\[
\vdash \text{HA}_b (MN)
\]

Case: $x : A$ : $A + x : A + x : A + A$

Stuck: Can't apply $\text{HA}_b$ to $M$.

TS: $\text{HA}_b (x : A, M)$.

Suffices TS:

if $\text{HA}_b (A)$ then $\text{HA}_b (x : A, M)$.

Suppose $\text{HA}_b (A)$, know that

$(\lambda x. A, M) : \text{HA}_b (A, M) 
\Rightarrow (\text{HA}_b (A, M) x : x : A$.

Idea: Say something about open terms

as "mappings" given by subset.

Note: $\Rightarrow$ for $\Rightarrow$.

"not --- not".

Strengthen the TS even further:

If $\Gamma + M : A$ then

for all $\forall y : \text{MT}_b A$ (closing subset)

if $\text{HA}_b (A)$, then $\text{HA}_b (\forall (M))$.

Case: $\text{HA}_b (\forall (M))$.

Proof:

Var: $\Gamma, x : A + x : A$, $\forall (x) = \forall (x) : A$.

Our assumption $\text{HA}_b (\forall (x))$.

Case: Immediate

App: By TS, $\text{HA}_b (M), \text{HA}_b (\forall (x))$

\[
\vdash \text{HA}_b (\forall (M)), \quad \text{and} \quad MN = MN.
\]
\[ \frac{\Gamma, x : \text{A} \vdash \text{M} : \text{B}}{\Gamma \vdash \text{\lambda}x. \text{A} \cdot \text{M} : \text{A} \to \text{A}} \]

Suppose \( \text{HT}_\text{A}(\text{M}) \). \( \text{TS} : \text{HT}_\text{A} \to \text{HT}_\text{B} (\lambda x : \text{A} \cdot \text{M}) \).

Suppose \( \text{HT}_\text{B}(\text{N}) \). \( \text{TS} : \text{HT}_\text{B} ((\lambda x : \text{A} \cdot \text{M}) \text{N}) \).

By \( \text{IH} \), \( \text{HT}_\text{B}(\text{\}[\text{N}/x]\text{\hat{M}}) \) because

\[ \text{HT}_\text{B}(\lambda x : \text{A} \cdot (\text{\}[\text{N}/x]\text{\hat{M}})) \text{, i.e., } \text{HT}_\text{B}(\gamma'(\text{M})) \]

Lemma ("head expansion"). \( \text{If } \text{HT}_\text{A}(\text{M}') \) and \( \text{M} \to \text{M}' \), then \( \text{HT}_\text{A}(\text{M}) \).

It determines matters.

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

\[ \square \]
Last time

We tried showing
\[ \text{Ann: } \frac{\text{H}_M \vdash A, \text{ then } M \text{ terms}}{\text{H}_M : A : M \text{ terms}}. \]

We introduced hereditary termination to accomplish this. We defined:
- \( \text{H}_M (M) \) iff \( M \text{ terms} \)
- \( \text{H}_A \rightarrow B (M) \) iff \( \forall N \text{ if } \text{H}_A (MN) \text{ then } \text{H}_B (MN) \)

Instead of showing \( M \text{ terms} \), we show \( \text{H}_M (M) \)

And then

\[ \text{Ann: } \frac{\text{H}_A (M)}{\text{H}_M (M) \text{ terms}}. \]

We had to generalize to \( \text{H}_A (\Gamma) \):
\[ \text{Ann: } \frac{\Gamma \vdash M : A \text{ and } \text{H}_A (\Gamma), \text{ then } \text{H}_A (\Gamma (M))}. \]

Here
\[ \text{H}_A (\Gamma) \text{ iff } \forall x : A \in \Gamma, \text{H}_A (\Gamma (x)), \text{ and } \text{ dom } \Gamma = \text{ dom } \Gamma. \]

Remark: \( \text{H}_A (\varnothing), \varnothing (M) = M \).

Lemma (head expansion):
\[ \text{If } \text{H}_A (M^i) \text{ and } M \rightarrow M' \text{ then } \text{H}_A (M) \]

Proof: By induction on the structure of \( A \).

Case \( A = A \rightarrow A' \): 
- Suppose \( \text{H}_A (M^i) \text{ and } M \rightarrow M' \).
- \( \text{TS: } \text{H}_A (M^i) \rightarrow \text{H}_A (M) \).
- Suppose \( \text{H}_A (N) \), \( \text{TS: } \text{H}_A (MN) \).

By assumption, \( \text{H}_A (MN) \) to \( \text{H}_A (MN) \).

This works exactly because we have head reduction \( \frac{M \rightarrow M'}{MN \rightarrow MN} \).

This is why head reduction is the way it is.
If $HT_A(M)$, then $M$ term.

If: By induction on $A$.

Case: $A=b$. Immediate.

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

What to do?

1. Give up! Empirical properties only are observables. Case type. Change the theorem to: If $M : b$, then $M$ term.

2. Change the definition of $HT_A(M)$ to also insist $M$ term. Then $\rightarrow$ 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 way of defining $HT_A(M)$:

- $HT_b(M)$ iff $M \rightarrow^* c$ (no real change).
- $HT_{A \rightarrow A'}(M)$ iff $M \rightarrow^* \lambda x : A. M_2$ and for all $M$ s.t. $HT_A(M')$, we have $HT_A(M' \beta x M_2)$.

This is called the method of canonical forms:

- $HT_A(M)$ iff $M \rightarrow^* \text{Vimal} \rightarrow^* c_\alpha$, $c_\alpha (\forall x. A. M)$.

Where $HT_b(c) \iff M : c_\alpha$ and $HT_{A \rightarrow A'}(\lambda x : A. M)$.
Then the intro rules become easy:

\[
\frac{
\Gamma, x : A_1, \vdash M_1 : A_2 \\
\Gamma, x_1 : A_1, M_1 : A_2}
}{\Gamma, x : A_1 \vdash \lambda x_1.M_1 : \forall x_1. (A_2) \\
\Gamma, x_1 : A_1, M_1 : A_2}
\]

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 

\( \langle \lambda \rangle \) 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! \( F(M) \) you need to evaluate. (\( M \) regardless of whether you evaluate it)

\[
\begin{align*}
HV^{CBV}(x : A, M) & \iff HT_{A_1}^+(M_{1}) \otimes HT_{A_2}^+(\langle M_{1}, x \rangle M) \\
HV^{CBV}(x : A, M) & \iff HV_{A_1}(M_{1}) \otimes HT_{A_2}^+(\langle M_{1}, x \rangle M).
\end{align*}
\]

Now consider what happens when you try to show \( CBR \):

Claim: \( \Gamma, M : A \) and \( HV_{A_1}(Y) \), then \( HT_{A_1}^+(\langle Y \rangle M) \).

Thus \( HT_{A_2}^+(M) \) ad \( HT_{A_2}^-(N) \). Is: \( HT_{A_2}^+(N) \).
$N \rightarrow^{*} V$ and $H_{VA}^1 (V)$. But here you can't just give up: you must show this.

To summarize:

1) If $P \vdash M : A$ and $HTp (V)$, then $HT (V (M))$.
   The formalism (syntax) "P \vdash M : A" has a semantics if $HT (V)$, then $HT (V (M))$.

Notation:
1) A type (grammar) $M \in A$ iff $HT (M)$
2) This is a behavioural specification. $M : A$ has no such meaning: you just derived it from a bunch of rules. You can, however, make the implications we did.
3) $\Gamma \Rightarrow M \in A$ means $\forall \xi \Gamma$ implies $\xi (M) \in A$.

The fundamental theorem then says: true implies true.

Safety: $\Gamma \vdash_M A$, then $\Gamma \Rightarrow M : A$.

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 as a means of accessing the truth, and as such is never canonical.

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

$\text{M1 name : } \lambda x : A . M \equiv \lambda x : A . M$
Changing gears.

Normalization

Interpret $M + M : \beta$ as a mapping on open terms (not closed terms).

Now the notion of behaviour is given by

$M \text{ mmp } N$ means $M \to^\beta N$.

$Lx : A. M \Downarrow Lx : A. M'$

meaning

$M \to^\beta M'$

$Lx : A. M \to Lx : A. M'$

Reduction is restricted to defined open terms.

$N \text{ mmp } \beta$ iff $N \to^\beta$.

Ep: ind characterize $N \text{ mmp }$.

Hint: If $N \equiv M : A$, then $M \text{ mmp } N \equiv N$.

Pf: by induction on typing.

We again get stuck at application.

$M \text{ mmp } N \text{ mmp }$.

TS: $M \text{ mmp } N \text{ mmp }$?

Stuck as usual.
Defining Imaginary Numbers

HNA (M) = IM

Terminology:
- A. H. A.
- A. P. A.
- H. M. A.
- H. P. A.

Analogous method (second method)
- Analysis

HN A (M)

2. If HNA (M) is a mapping,
   then HNA (M) simplifies by common

3. If T-M.A. HNA (M)
   then HNA (M) simplifies by common

4. If T-M.A. HNA (M) simplifies by common

5. If T-M.A. HNA (M)
   then HNA (M) simplifies by common

Comments on mapping:
- From HNA (M)
- To HNA (M)

Use of HNA (M) to introduce the concept of

Prove: What can be done with the concept of

HNA (M)
We tried

\[ \text{M: } \Gamma + M: A \implies M \text{ means } X \text{ can't } \]

\[ \text{then try } \]

\[ \text{M: } \Gamma + M: A \implies A \text{ and } \Delta + \forall \phi \quad \text{ and } \]

\[ \text{HN}_A(\Delta) \text{ then } \text{HN}_A(\Delta(M)) \]

Our first attempt at \( \text{HN}_A(M) \) will be to assume:

\[ \text{HN}_A(M) \quad \text{ i.e. } \quad \Delta \vdash M: A \]

\[ \text{HN}_B(M) \quad \text{ i.e. } \quad M \text{ means } \text{HN}_B(\Delta(M)) \]

\[ \text{HN}_A(\Delta(M)) \quad \text{ i.e. } \quad \text{HN}_A(M) \text{ implies } \text{HN}_A(\Delta(M)) \]

Lemma

1. \( \text{HN}_A(\Delta(M)) \) — i.e., \( \text{HN}_A(\Delta(M)) \)

2. \( \text{HN}_A(M) \) implies \( M \) means

Proof:

1. \( \Delta + \forall \phi \quad \text{Then } \exists \gamma(\Delta) = \gamma(\Delta) \text{ by assumption } \text{HN}_A(\Delta(M)) \)

2. \( \gamma(\Delta) = \exists \gamma(\Delta) \quad \text{Now} \)

\[ \Delta + M: A \implies B \quad \Gamma + \forall \phi \quad \text{HN}(\Delta(M)) \]

\[ \implies \text{HN}_A(M) \]

\[ \implies \text{HN}_B(\Delta(M)) \]

\[ \implies \text{HN}_B(M) \]
\[ \Gamma, \alpha : A \vdash M : B \quad \text{IH: If } \Gamma \vdash N : A \quad \therefore \quad \text{TS: } HN^A_B (\alpha : A, M) \]

Suppose \( HN^A_B (N) \). \( \text{TS: } HN^A_B (\alpha : A, M) \)

Result follows by head expansion. D.

So still need to show: D.

Lemma: 3. If \( M \rightarrow M' \) and \( HN^A_B (M') \), then \( HN^A_B (M) \).

\text{Pf:}\quad \text{Proc depends on } A.

\text{1. } \alpha = b : \quad \text{TS: } HN^\alpha_B x : b (\alpha). \ i.e., x \text{ norm } B.

\text{2. } A = A_1 \rightarrow A_2 : \quad \text{Suppose } HN^\alpha_{A_1} A_2 (M'). \quad \text{TS: } HN^\alpha_{A_2} (M).

\text{Suppose: } HN^\alpha_{A_2} (M_2). \quad \text{TS: } HN^\alpha_{A_2} (M(M_2)).

\text{But that follows by the IH, because the type gets smaller.}\]

\text{Pf of (1) and (2): By ind on } A.

1. \( \alpha = b \): \( \text{TS: } HN^\alpha_B x : b (\alpha) \). I.e., \( x \text{ norm } B \)

2. \( A = A_1 \rightarrow A_2 \): \( \text{TS: } HN^\alpha_{A_1} A_2 (\alpha) \).

\text{Specifically: } HN_{A_1}^{\alpha_2} A_2 (M_1). \quad \text{TS: } HN_{A_2}^{\alpha_1} A_2 (x(M_2)).

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

By IH(2), \( M \text{ norm } B \). But you're stuck, so we need to strengthen the IH. See 17.
2. \( A \rightarrow b: \text{spec } HN^b(b) \), \( TS: M_{\alpha_{M}} \). \\
\( A = \alpha_{A} \rightarrow \beta_{A}: \text{spec } HN^\beta_{\alpha_{A}}(M) \), \( TS: M_{\alpha_{M}} \).

But we don't have any \( A \) to apply this to, so we're stuck.

But \( \text{IH}_{A}^{1} \), \( \text{CMN}_{A}^{x} \), (Dubius step but ignore this), \( \text{HN}_{A_{n}}(M_{\alpha_{A}}) \), \( \text{by } \text{IH}_{A}^{1}, M_{\alpha_{M}} \).

Check this inference carefully.

Dubious because I might be empty or never have any variables of type \( A \).

The standard trick for decades was to use "induced contexts" for containing co-many uses for co-many steps, and then you show the compatible property where if \( \text{Go} + M_{A} = E_{R} \), then \( P: A \).

\[ \text{Strengthen (the old way)} \]

If \( R_{A}: A_{1} \rightarrow A_{n} \rightarrow \cdots \rightarrow A_{n-1} \rightarrow A \) slightly
and \( M_{\alpha_{M}} \) grows \( \cdots \) the norm grows
\[ \text{HN}_{A_{n}}(\cdots M_{\alpha_{M}}). \]

\[ \text{See (9) for clearer form.} \]

Patch up pf qf 1.

\( A = b: \text{spec } HN^b_{\alpha_{b}}(M, M, M_{\alpha_{M}}) \).

\( b \in M_{\alpha_{M}} \).

\( M_{\alpha_{M}} \) normalizes to \( M_{\alpha_{b}} \).

\( M_{\alpha_{M}} \rightarrow M_{\alpha_{b}} \rightarrow M_{\alpha_{b}}. \)

(Just normalize the \( M_{i} \) to normalize \( M_{\alpha_{M}} \).)

\( A = \alpha_{A} \rightarrow \beta_{A}: \text{spec } HN^\beta_{\alpha_{A}}(M_{\alpha_{M}}) \).

\( \text{HN}_{A_{n}}(M_{\alpha_{M}}) \), \( TS: HN^\beta_{A_{n}}(M_{\alpha_{M}}) \).

\( \text{IH}_{(2)} \) \( P_{\alpha_{M}} \), \( M_{\alpha_{M}} \) normal. So by \( \text{IH}_{(1)} \)
A better approach than Bo:

Kelly: Kripke semantics and presheaves.

With our computer hacker hat on, you realize that what you really want to do is allocate new variables for what you want smaller.

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

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

So we change the definition of \(HN\):

Let \(\Delta' \supset \Delta\) iff \(\Delta'\) extends \(\Delta\) a "future world" (from \(\Delta\) refl.)

Then

\[HN(A_1 \rightarrow A_2)(M) \iff (\Delta' \supset \Delta \text{ and } HN(A_1)(M)) \text{ and } \Delta' \text{ satisfies } \Delta, \text{ and } \Delta \text{ is reflexive} \]

Fact (monotonicity/functionality):

\[\text{If } HN(A_1)(M) \text{ and } \Delta' \supset \Delta, \text{ then } HN(A_2)(M).\]

Now you can go through the proof and allocate new \(A\) at \(x\), by IH, \(HN(A_1)(M)\).
Lemmas:

1) \[ \text{If } E : C \rightarrow A \text{ and } E' : M \rightarrow \text{ then } H^A_{\text{norm}}(E \langle M \rangle) \]

2) \[ \text{If } H^A_{\text{norm}}(M) \text{ then } M \text{ norm} \]

Rank about strong normalization:

There is an obvious in the literature about strong normalization, SNP.

Roth claims that the only important thing about SNP is

"Never important in itself!"

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

no α reduction sequence:

\( M = M_0 \rightarrow \beta_1 \rightarrow M_2 \rightarrow \beta_2 \rightarrow \ldots \)

what this means is that the principle of transfinite induction on reduction is valid.

\( \forall \beta \exists (M_0, M \rightarrow \beta, \beta_1 \rightarrow \beta) = \text{valid} \)

so no inf net sequence = \text{TIR is valid.}

This follows from König's Lemma.

Each \( M \) has finitely many β reduces

\( M_0, \ldots, M_k \)

finite fan out, all branches

\( \beta_1 \rightarrow \beta_2 \rightarrow \ldots \rightarrow \beta_k \)

are finite by SN. So

tree finite by König.

This is the only good reason you have.

By \( \text{E:C and } A \rightarrow \text{part (1) of the lemma on (B)} \),

\( \Delta \vdash c, \ \Delta \vdash \text{E}(\rightarrow A) \)

[ chtt - 218. slack. com ]
Last time:

\[ \text{FTLR : if } \Gamma + \Gamma : A \text{ and } \chi : A \to P, \text{ then } \]
\[ \text{HN}_0(\Delta) \text{ implies } \text{HN}_0^{\Delta}(\chi(M)). \]

Case 1: of \( \Gamma + \Gamma : A \), then \( \text{M norm}_P \)

Structure "\( \Gamma + \Gamma : A \)" determines behaviour. "\( \text{HN}_0^{\Delta}(\chi(M)) \)."

To show (2), you need to show that \( \text{HN}_0^{\Delta}(\chi) \), and to do so, you meet
show that

\[ \Delta + x : A, \text{ then } \text{HN}_0^\Delta(x). \]

But you get this by using

\[ \text{if } \xi \text{ norm}_P, \text{ then } \text{HN}_0^{\Delta}(\xi(x)) \]

Not: \( x : M \Gamma \) is \( \text{HN} \) when \( M \in \text{HN} \)

\[ \text{Not: } \text{HN}_{\text{not}}^\Delta(M) \text{ implies } \text{M norm}_P \]

Idea: Allocate a fresh variable at type \( A \), and consider \( M \xi \) at \( A_2 \).

\[ \begin{align*}
\text{M norm}_P & \quad \text{M norm}_P \\
\text{Claim: Kripke logical rel.'s are the} & \quad \text{means for talking about state in Pl.} \\
\text{When allocating a fresh variable,} & \\
\text{you move from } \Delta & \text{ to } \Delta + x : A, \text{ where } x \notin \Delta \\
\text{But you need a stability property,} & \\
\text{which tells you that moving to a} & \\
\text{future world does not affect} & \\
\text{hereditary normalization.} & \\
\end{align*} \]

\( \text{HN}_0 \) : Worlds \( \to \) Rel (\( \cong \) sets) \( \Delta \) ordered
Exercise: (Closed and open — do both)

Add "negative" products. Defined by:
\[ \langle M_1, M_2 \rangle \mapsto \text{elim form} \]
\[ M_1 \cdot 1, M_2 \cdot 2 \mapsto \text{projections} \]
\[ \langle M_1, M_2 \rangle \cdot 1 \mapsto M_1, \]
\[ \cdot 2 \mapsto M_2. \]

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

2) \( HN_{A_1 \times A_2} (M) \) — reprove norm.

Positive Types
(Exercise: do positive products)
\[
\frac{\text{let } \langle x, y \rangle \in M \in N}{\text{let } \langle x, y \rangle \in N_{x,y}}
\]
\[
\text{matching.}
\]

Prime example: booleans.
\( \text{true, false are values/intro/elim forms.} \)
\[
\frac{\text{if } \langle M', P, Q \rangle}{\text{elim}}
\]
\[
E := \quad \text{if}(E, P, Q).
\]
\[
\frac{\text{true } \cdot P \mapsto P}{A}
\]
\[
\frac{\text{false } \cdot P \mapsto Q}{A}
\]
\[
\frac{\text{true } \cdot (M', P, Q) \mapsto P}{A}
\]
\[
\frac{\text{false } \cdot (M', P, Q) \mapsto Q}{A}
\]
\[
\frac{\text{true } \cdot (M', P, Q) \mapsto \text{true}}{A}
\]
\[
\frac{\text{false } \cdot (M', P, Q) \mapsto \text{false}}{A}
\]

Prime: booleans.
\( \frac{\text{true } \cdot \text{false } \mapsto \text{true}}{A} \)
\( \frac{\text{true } \cdot \text{false } \mapsto \text{false}}{A} \)
\( \frac{\text{true } \cdot \text{false } \mapsto \text{true}}{A} \)
\( \frac{\text{true } \cdot \text{false } \mapsto \text{false}}{A} \)
\( \frac{\text{true } \cdot \text{false } \mapsto \text{true}}{A} \)
\( \frac{\text{true } \cdot \text{false } \mapsto \text{false}}{A} \)
\( \frac{\text{true } \cdot \text{false } \mapsto \text{true}}{A} \)
\( \frac{\text{true } \cdot \text{false } \mapsto \text{false}}{A} \)

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

If \( A \) is a "recursor", which witnesses that \( \text{bool} \) is inductively defined by
\[
\text{true, false.}
\]
Let's revisit the termination proof for closed terms. This should go through straightforwardly:

\[ HT_{\text{bool}}(M) \text{ iff either } M \rightarrow^* \text{ true or } M \rightarrow^* \text{ false } \]

Remark: for function type, you can give a proof by structuring term. But with your unary positive type, you have no choice but to give a positive characterization of \( HT_{\text{bool}} \).

Easy: \( \text{if } HT_{\text{bool}}(M), \text{ then } M \text{ term} \).

Easy: Head expansion.

ETLR: \( \text{if } \Gamma \vdash M : A \text{ and } HT_A(y), \text{ then } \{ \text{ closed } \}

Con: \( \text{if } M : A, \text{ then } M \text{ term} \).

Exercise: do \( HT \) for positive products.

\[ HT_A(x, y)(M) \text{ iff } \begin{cases} M \rightarrow^* y & \text{ if } \Gamma \vdash M : A \\ \text{ and } \Gamma \vdash H_T \end{cases} \]

Consider: Could it somehow give a negative characterization of \( \text{bool} \)? That is, defined in terms of \( \neg \). If it could, I'd want to say something along the lines of

\[ HT_{\text{bool}}(M) \text{ iff } \begin{cases} 1. \text{ HT}_A(P) \\ 2. \text{ HT}_A(Q) \end{cases} \text{ then HT}_A(\text{if}(M; P, Q)). \]

What's wrong with this that it violates" cascade definition by recursion on the type? We define \( HT_{\text{bool}} \) in terms..."
We're characterizing the behaviour of bool
in terms of arbitrary types A.

Vertigo

bool = ∀A. A → A → A

are the Church booleans.

We claim 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 \( \Delta \vdash A \rightarrow \Gamma' \), then

\( \text{HN}^A_{\Gamma'}(Y) \) implies \( \text{HN}^\circ_A(\hat{\Gamma}(M)) \).

How do we define \( \text{HN}^\circ_A(M) \)? (This won't
generalize nicely so consider next.

Cannot:

\( M \rightarrow^* \text{true on } M \rightarrow^* \text{here!} \)
Because \( M \) is open and could be "x",
or "x \text{N}" or "if (x \text{N}; P; Q)" or...

Constraints: We must validate:

1) \( \text{HT}^A \) (true) and \( \text{HN}^A \) (false)

2) if \( \text{HN}^A (A) \) and \( \text{HN}^A (B) \) and \( \text{HN}^A (Q) \),
by structural

3) \( \text{HN}^A (\hat{\Gamma} (M; P; Q)) \).

What do we know about \( \hat{\Gamma} \)? That \( \hat{\Gamma} \) nom.

Fact: if \( M \) nom, then \( M \rightarrow^* \text{N} \rightarrow^* \)

You're faced to do the head-\( \beta \)-reduction. \( \text{N} \) nom.

It reduces to something that is head wired, i.e.,
in nuf.
Define $HN^A(M)$ iff $M$ nonp.

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

By definition, $M$ nonp.

Lemma (head exp.) ...

Lemma (work home):
1) If $HN^A(M)$, then $M$ nonp.
2) If $E$ nonp, then $HN^A(E^2B^3)$.

By Lemma on (1) and (3), we have $F$ nonp.

$M$ nonp, so $SN$, $N$ true and $M ightarrow N$.

We proceed by case analysis on $N$.

$N$ = true:

TS: $HN^A(\text{if}(\hat{M}; \hat{P}; \hat{Q}))$

then

true

So $\hat{P}$, which we know $HN^A(\hat{P})$ by (2).

So we're done by head exp.

$N$ = false: symmetric.

Otherwise: What do we know?

$N$ is in head and is of type bool. So it must be true.

This is the current state: $[N = E^x M \wedge E \text{ nonp}]$

part of today's lecture: $HN^A(E^x M)$ by Lemma 2)

$HN^A(\text{if}(E^x M; \hat{P}; \hat{Q}))$

Then use head expansion.
Today: Finish our discussion of picture types.

Last time we discussed booleans:

  - Doi' algebra:
  2. Open interp - variables are undetermined.

Define $HT\text{bool}(M)$ iff $M \rightarrow \text{true}$. $\Delta$.

$\Delta$ is the "state" - available undetermined.

$HT\text{bool}(M)$ iff $M \text{ norm}$. (1)

The important point is that there are infinitely many booleans in the open case and yet you can just throw $FTLR$ with (i).

Before look at the logical order of ideas:

- The open case was:
  1. Head exp
  2. $F\text{TLR}$
  3. Workhorse lemma - inserted by Bob this week

After look at the order changes:

- 1. Head exp
- 2. Workhorse lemma (needed to prove)
- 3. $F\text{TLR}$

Let's cook or

Sums more generally

Want the types:

0, $a \text{ and void}$

A $\times$ B

Formal typing:

$\text{M} : \text{void}$

$\text{P} \land \text{M} : \text{A}$

$\text{P} \lor \text{M} : \text{B}$

$\text{P} \& \text{M} : \text{A} \land \text{B}$

If you have $\text{P}$ then $\text{M} \text{ void}$ is necessary.

If you have $\text{P} \land \text{M} : \text{A}$ then $\text{M} : \text{void}$ is necessary.

If you have $\text{P} \lor \text{M} : \text{B}$ then $\text{M} : \text{void}$ is necessary.

If you have $\text{P} \& \text{M} : \text{A} \land \text{B}$ then $\text{M} : \text{void}$ is necessary.
1. Closed setting: straightforward

\[ \text{HT\text{\small valid}}(M) \iff \text{Claim} \]
\[ \text{iff} \quad \text{-HT\text{\small valid}}(M) \quad \text{any M} \]

For \( \exists \text{HT\text{\small valid}}(M) \iff \) either \( M \rightarrow 1 \rightarrow N \) or \( M \rightarrow 2 \rightarrow N \) and \( \text{HT\text{\small valid}}(N) \).

Check head exp: FTLR

If you were to attempt a negative definition, you'd run into the same problem as before:

\[ \forall X. (X \rightarrow X) \rightarrow (B \rightarrow X) \rightarrow X \]

The whole inductive character of our enterprise breaks down. Rmk: It comes from considering the negative form, and this is how you know to be unencodings in general.

2. Open case:
We'll move in less recapitulative fashion

\[ \text{HT\text{\small valid}}_{A \lor B}(M) \iff ? \]

Considering \( \& \) will tell us what we need. By \( \text{IF} \), we'll know:

1. \( \text{HT\text{\small valid}}_{A \lor B}(M) \)

2. If \( \text{HN\text{\small valid}}_{A}(N) \) then \( \text{HN\text{\small valid}}_{A}(N/x)P \)
3. If \( \text{HN\text{\small valid}}_{B}(N) \) then \( \text{HN\text{\small valid}}_{B}(N/NQ) \)

Then are wrong, if \( \& \) they do not respect the fundamental property of maps, namely, that they are invariants under allocation.

So insert
So, switch.

2) If \( A' \neq A \), \( HN^A(N) \) then \( HN^A_c([N/x]P) \)

TS: \( HN^A_c(\text{case}) \).

Think of how the proof may go:

Proceed by case analysis on \( A' \):

1) \( M \mapsto M' \): head reduction/head up

2) \( M \rightarrow P \), e.g., \( \text{bc } \tilde{M} \text{ so a var.

3) }

Proceed by case analysis on \( A' \):

1) \( N = 1 \cdot N' \): Case \( \rightarrow [N'/x]P \), and we
want this to be \( HN^A_c \). But
then we want \( N' \) to be \( HN^A_c \).

And then you get what you
want by head up

2) \( N = 2 \cdot N' \): Symmetrical.

3) \( N = \xi \xi' \): Then

\( \text{case } \xi \xi' \rightarrow P \rightarrow \Pi x \rightarrow Q \)

Let \( \xi' = \text{core } \xi' \xi' \)

Then apply the workhouse lemma and

done.

\( HN^A_{\text{top}}(N) \) if

1) \( M \) normal

and

2) \( \text{if } M \rightarrow 1 \cdot N \), then \( HN^A(N) \), and

3) \( \text{if } M \rightarrow 2 \cdot N \), then \( HN^A(N) \).
Rmk: Now we're in a perfect position to be able to do positive products let's be \( M \) in \( N \) let's be \( A \) in \( N \).

**Natural Numbers:** Gödel's \( T \).

\[
\begin{align*}
A & : \text{nat} / A, \rightarrow A \times A \\
M & : \text{zero} / \text{succ}(M) \\
\text{rec}^A (P, x, Q)(M) & : \text{PRPR}
\end{align*}
\]

1. If \( M : A \), then \( M \) temp. (If you want, you can extract.
2. If \( P \vdash M : A \), then \( M \) temp.

We're going to show a stronger prop than HTM.

If \( P \vdash M : A \) and \( HT^P (\vec{p}) \), then \( HT^A (\vec{f}(M)) \).

HT\text{nat} (M) if \( ?? \). This should imply \( M \) temp.

We're talking about positive types so you do the obvious thing...
HT(m) if either $M \rightarrow^{\infty} 0$ or $M \rightarrow^{\infty} \text{ since } (N)$ and $HT_{\text{max}}(N)$.

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

You have "vertical" HT induction defined on a structure of $A$.

This is a "horizontal" one at the level of $n$.

Define $HT_{\text{max}}(\cdot)$ to be the strongest $P$ on map $s. t. f$ if $M \rightarrow^{\infty} y$, then $P(M)$

\[ HT_{\text{max}}(S) \text{ or } P(N) \]

Alternatively, you could say

\[ HT_{\text{max}} = \bigcup_{\text{max}} F^{(\infty)}(\phi) \]

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

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 "coated", you'll find a different formulation acceptable.

You can't deny you believe in set theory but not "not".

In the rule of $\cup$, you're going to need to carry the $RDL$ and write some element of the form: $\text{formulas,}$

\[ P \cup R \cup C \]

\[ R \cup C \rightarrow P \cup A \cup C \]

\[ P, \text{max } \rightarrow (P \cup A) \cup C \]
The point is that that can be proven by the hoary inductive principle.

The P of (\(P\)) is

\[ P(\mathit{N}) \mathcal{\#} \mathit{HTC} (\tilde{\mathit{rec}}(\mathit{W})) \]

**Remark:** \(\widetilde{\chi}\) is not mathematical induction! You're reasoning about a program's correctness.

If you accept \(\varphi\), then you're accepting the consistency of \(\mathsf{PA}\).

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

\[ \mathit{HN}_{\alpha} (\mathit{M}) \mathcal{\#} \mathit{M}_{\alpha_{\varphi}} \]

and

1. if \(\mathit{M}_{\alpha_{\varphi}}\) is true then \(\mathit{true}\).
2. if \(\mathit{M}_{\alpha_{\varphi}}\) is true then \(\mathit{HN}_{\alpha_{\varphi}} (\mathit{M})\).

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 Bostrom's of the world; obviously one of my favourite computer scientists." — Peter W. 31
Neqt

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

Let's talk about:

Equality

Formal Type Theory

"Definitional equality" (forget that definition, it has no meaning)!

Introduce a judgment

\[ \Gamma \vdash M \equiv M' : A \]

\( \equiv \) is refl., trans., symm., congr., & recursive. Needs
\( \beta \)-principle.

All of the virtues of dep \( \Pi \Pi \) can be formalized in dep \( \Pi \Pi \) over \( \equiv \).

This is where you encode computation in \( \Pi \Pi \), except that it's not directed!

\[
\begin{align*}
\Gamma \vdash M \equiv N : A \rightarrow B \\
\Gamma \vdash N \equiv N' : A \\
\Gamma ; M ; N = M' ; N : A \rightarrow B \\
\Gamma ; (\lambda x : A . M) ; N \equiv (\lambda x : A . M) ; N \\
\end{align*}
\]

You get them hard issues like

\[ \lambda x : A . x \neq \lambda y : B . y \]

\[ 0 + x = x \]

[Note: you may have used \( f(x) \) for \( f(x) = f(y) \) in this case.]
"exact equality" - terminology by Carlo A and R. H.

(A type)

\[ M \equiv A \]
\[ M \equiv M' \subseteq A \]
\[ M \equiv M' \subseteq A \]

Recall examples from first lecture.

Maps respect exact equality!

\[ x : A \Rightarrow P \Rightarrow Q \in B \]

mean

if \[ M \equiv M' \subseteq A \] then

\[ (M/x)P \equiv [M'/x]P' \subseteq B \]

Open terms are functions of their variables.

Then the fundamental theorem tells us

\[ \text{If } P \Rightarrow M \equiv M' : A, \text{ then } P \Rightarrow M \equiv M' : A. \]

\[ \text{33} \]
So far, we've considered
\[
\text{not bool } 0 + 1 \rightarrow \text{ neg}
\]
inductive, pro...

(won't do coinductive types — neg).

Today: polymorphism: \(V_x F_x\) — second order guess.

This is type/prop quantification: 
\[\forall x. A \quad \exists x. A\]
union types

We'll begin by a formal definition:

\[
\Gamma, x: A \vdash x: A
\]
\[
\Gamma \vdash A \rightarrow A_1 \quad \forall x. A \rightarrow A_2
\]

\[
\Gamma, x:A, M_2:A_2 \vdash \Gamma, M:A_1 \rightarrow A_2
\]
\[
\Gamma, x:A, M_2:A_2 \vdash \Gamma, M:A
\]

\[
\Gamma, X:M:A \quad \Gamma, X:M:A
\]

\[
\Gamma \vdash M:VX.A \quad \Gamma \vdash M:VX.A
\]

\[
\Gamma \vdash M:VX.A \quad \Gamma \vdash M:VX.A
\]

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

\[
\text{Var: } \text{ a finite set of } x
\]
\[
\text{Var: } \text{ a finite set of } x
\]

Erasure. So far we've written

\[M:A \text{ implies M terms.}\]
But really, we mean

M ⇒ A implies IM1 terms,

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

Similarly, we really mean

\( \vdash M \vdash A \) (and \( \vdash H \vdash (\lambda) \))

\( \vdash H \vdash (\lambda (1M)) \)

Idea: the semantics (i.e. \( H \vdash \)) is
defined on erased terms.

In our polymorphic setting, there
are two reasonable approaches.

(I)

\[ M \vdash M \square = \square \]

"full erasure" / "type assignment"

\( 1M[A]\vdash = 1M \]

common in the intuitionistic school,
and even used in ML, but has problems
Unsoundness in ML polymorphism due to turning value \( 1M \) to nominal \( A \)

(II)

\[ M \vdash M \square = T \quad 1M \]

- delays

\( 1M[A]\vdash = 1M \) (\( \vdash \)) - activates

Our goal is to show \( \text{II} \) doesn't really
matter, which erasure style you use, but
\( \text{REL} \text{II} \) likes II so we'll do that? \( 1M \Rightarrow 1M \)

Rite of Passage for any PL person:

a) Try to prove \( \text{I} \) 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 first-order math. And Gödel's Incompleteness 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:

$HT_T(A(M))$ iff $M \leftrightarrow \text{true}$ or $\top$

$(\text{say})$ $HT_A(M)$ iff $HT_A(M_1) \lor HT_A(M(M_1))$

But what do you do at $\forall X.A$ or $\exists X$?
Perhaps on a first cut stick to closed types.

$HT_{\forall X.A}(M)$ iff $(\forall \text{closed } B, \quad HT_{B/X}A(M_{B}))$

"generics" (you don't know what you're doing)

Why does Russell dismiss this?
Rmk: you don't need $HT$, in this case.
Rmk: Be careful: this is supposed to be defined on closed terms, in which case you wouldn't write $M_{B/X}$, 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.

$\forall X. X \rightarrow X \rightarrow (\forall X. X \rightarrow X) \rightarrow (\forall X. X \rightarrow X)$

Your rest failed more: 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 successor!

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

\[ HT_A(M) \] iff \[ HT_{\lambda x:B} A(x) \] (\Theta)

\[ \text{open A: } \Theta \rightarrow A \rightarrow \Theta \]

\[ \text{closing maps } \delta : \Theta \]

\[ HT_x(M)(\delta : \Theta) \text{ iff } HT_{\theta(x)}(M) \]

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

\[ HT_A(M(\delta))(\delta, x:\delta B : \Theta) X \]

Still ill-defined!

Try: Define \( HT \) on the skeleton of the type!

\[ HT_A(M) \]

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

\[ HT_x(M)(\delta, x : \delta A : \Theta) \text{ iff } \]

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

\[ HT_A(M(B))(\delta, x : \delta B : \Theta) \eta \]

Remarks: This is only a restatement of the previous definition, except for the technical formulation.

Remark: The A is getting smaller!
This "standard interp" is what will hang you, because you could have $B = \text{true}$ and you would be stuck. What to do?

_Type Candidates!_

aka, admissible non-standard interps of types.

"possible types?"

*You need to accept comprehensive quantification over all types — to accept this proof. And the proof refers to consistency of $\text{FOArith}$, which log Gödel can't be done in $\text{FOL}$. So you need something else, and $\text{SOL}$ is this something else.*

\[
\text{HT}_x(M) \equiv \forall \eta (x/M) \\
\text{HT}_{x:A}(M) \equiv \forall \eta \forall \xi (x:A) \eta (\xi) \\
\text{HT}_{A}(M) \equiv \exists \eta \forall \xi (x:A) \eta (\xi) \Rightarrow (x \Rightarrow T).
\]

*Remark: We're not going mentioning $\text{HT}$ on the right! This — where we took $\text{HT}$ is one of the meatballs in the soup, but we don't name it.*

*Also: how can it be that you demand polymorphic types satisfy properties that aren't even expressible in the language?*
What are type candidates?

What makes it possible to do the proof of the FTR in the case of HT?

[What makes the proof work?]

- For of $HT_{A_0 \to A_1} = HT_{A_0} \to HT_{A_1}$
- $HT_{A_0 \times A_1} = HT_{A_0} \times HT_{A_1}$
- Head expansion!
- If $TM' = M(T)$ and $M = M'$, then $TM'$.

So possible types are sets of closed terms (erase terms) that are closed under head expansion.

2018-02-08

Girard's Methods aka Candidates

We want to define $HT_x$ for system F.

What do we do about $HT_{x_0}$?

Crucial idea: demand more of your programs than is evidently required, admitting non-standard types/type candidates.

Use 2nd order comprehension — sets of sets.

And allowing these unexpressible types is crucial for leveraging the full power of polymorphism, otherwise you end up with the impervious notion of generic. They are not just a technical trick!

$HT_{\hat{\lambda}}(M)[S_{\lambda}:D]$ [yes a candidate assignment] [the semantics of $\hat{\lambda} E$]
\[
\begin{align*}
HT_x(M)[\eta] & \text{ iff } \eta(x)(M) \\
HT_{\alpha_1}(M)[\eta] & \text{ iff } M, N \vdash x : \alpha_1 \text{ or } x
\end{align*}
\]

Good definition:

\[
\begin{align*}
HT_{\alpha_1, \alpha_2}(M)[\eta] & \text{ iff } HT_{\alpha_1}(M, \eta) \text{ simplifies } \eta \in HT_{\alpha_2}(M, M)[\eta] \\
HT_{\alpha_1, \alpha_2}(M)[\eta] & \text{ iff } \forall A, \forall x : \alpha_1, \forall \eta(x) \in HT_{\alpha_2}(M, M)[\eta]
\end{align*}
\]

\[
\begin{align*}
HT_{\alpha_1, \alpha_2}(M)[\eta] & \text{ iff } \forall A, \forall x : \alpha_1, \forall \eta(x) \in HT_{\alpha_2}(M, M)[\eta]
\end{align*}
\]

Cannot restrict attention to standard types! Cannot insist \( T = HT_{\alpha}(\cdot)[\delta; \eta] \).

- Collection of closed terms.

What is a type candidate? \( \mathcal{C} \) 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 candidate (i.e.) and \( T(M) \), then \( M \) terms.
3. If you had, e.g., recursion, then you need the admissibility of fixed point induction, see PFPL for details.

Remark: \( \dagger \) and \( \ddagger \) are formulated negatively which will impact what you can get out of the theorem below. To get a proper type instance, you need a positive form, and you can push this through.

The FTLR becomes more technical:

\[
\begin{align*}
\text{FTLR} & \text{ iff } \text{ Rank: If you want } HT(M)[\delta; \eta] \text{ simplifies } \\
& \quad \text{ HT(M) for all } M, \text{ then you need to use the pos formulation of } \eta
\end{align*}
\]

Otherwise, we restrict \( A = \alpha \).
If $\Theta F = M : A$, and
$s : \Theta$ closed types
$q : \Theta$ candidate assignment

\[ HT_\eta (Y)[s, q] \]

then
\[ HT_\eta (\gamma_1 \beta)(M)[s, q] . \]

\[ \text{Rank: \ } \beta(M) = \beta_1 M \quad \text{So we use} \quad \text{"There are those who teach, and those who use powerpoint." - RWH, on the use of chalk.} \]

as our conclusion instead.

**Lemma (Compositionality)**

We want to relate $HT\left[ M; \beta, \eta_1 \right]$ and $HT\beta(M)[s; x \mapsto A], \eta[x \mapsto HT(A)]$.

\[ HT\beta(M)[s; x \mapsto A], \eta \left[ x \mapsto HT(A) \right] \iff HT\beta(M)[s; x \mapsto A], \eta \left[ x \mapsto HT(A) \right] \]

**Proof:** By induction on the structure of $B$. \( \square \)

**Proof of FTCR:**

Case 1. $\Theta, x : \alpha \vdash M : B$.

Fix $\eta : \Theta, y : \beta \vdash HT_\eta (Y)[s, q]$.

**WTS:** $HT_{\alpha, \beta}(M)[s, q]$.

**Rank:** $\beta(M) = \lambda x . \gamma_1 M$.

Fix a closed $A$, candidate $T$.

**WTS:** $HT_B((\lambda x . \gamma_1 M)(t))[s; x \mapsto A], \eta[x \mapsto T]$.

by Lemma.

**WTS:** $HT_B(\gamma_1 M)[s', q']$.

by the IH, $HT_B(\gamma_1 M)[s', q']$. \( \square \)
2) $\forall x. B \vdash A \rightarrow \forall x. B$

Fix $s, \eta, \chi$. WTS: $HT_{\forall x. B}(\forall x.M(\eta))(s, \eta)$

By compositionality: $\forall x. B(\forall x.M(\eta))(s, \eta)$

By the IH,

$HT_{\forall x. B}(\forall x.M(\eta))(s, \eta)$

$\vdash HT_{\forall x. B}(\forall x.M(\eta))(s, \eta)$, and we're done. $\Box$

Case: $HT_{\forall x. B}(\forall x.M(\eta))(s, \eta)$ implies $M$-term

(To sort matters, first and foremost, first fix an $s$, and then fix $\chi$)

The critical point that makes this expression

in arithmetic is the set of sets. Thus

This formulation proves only what's 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:

Why 1. Equality format/semantic

2. Can do parametricity/data alone

by reifying the foregoing arguments
3. Props as types -> deep types
4. Deep types, CTT
5. HDTT types

data abstraction program terms

signature
signature

simple
simple

type t

val f : t -> t
val f = M₁
val f = M₂
val c : Nₐ
val c = N₈

M. t = t M₈ (t = R)
N. t = N₈ (t = R)

∀ t, (λ t. (t -> t) -> τ) \text{ type}

Client in polymorphim and the polymorphism ensures the implementation respect the relation relating the ref & hand implemented

\implies \text{ client can't tell the implementation apart}

\implies \text{ impl/expr independence}

Back to simple types: ans,

Could throw in 0, 1, x, +, but if you would

1. Formal structural TT

inductively defined i.e.

M : A interpreted \rightarrow a mapping over open terms.
Variables are indeterminates, calculation in "poly 2" open terms.

This is a proof theory a means of accessing the 1st truth.

*has no notion of programs*

The calculation done via P ≡ M2A A (where P ≡ M2A A)

≡ is the equality given by Gentzen's inversion principle: "elim

cancels intro".

T1: Semantical/declarational T1.

\[ \Gamma \vdash M : A, \quad \text{i.e.,} \quad \forall H, \forall \theta, (\Gamma) \text{ then } HT (\Gamma (\theta)). \]

These two are related by FTLR.

This T1 is a 'truth theory about programs

⇒ start w/ programs, type and spaces.

\[ \text{Defn of } \Gamma + \text{M} = \text{N} : A \]

\[ \Gamma + K : A \quad \Gamma + \text{M} = \text{N} : A \quad \Gamma + \text{M} \neq \text{N} : A \quad \Gamma + \text{M} = \text{P} : A \]

\[ \text{Lemma } \]

\[ (\forall) \quad \Gamma + \text{A} = \text{M} = \text{N} : \text{B} \]

\[ \Gamma + \text{Z} : A \quad \text{M} = \text{Z} : A, \quad \text{N} : A \quad \text{B} \]

\[ \Gamma + \text{M} = \text{Z} : A \rightarrow \text{B} \quad \Gamma + \text{N} : \text{A} \]

\[ \Gamma + \text{M} \neq \text{M} \neq \text{N} : \text{B} \]

\[ \Gamma + \text{Z} : \text{A} \rightarrow \text{M} : \text{B} \quad \Gamma + \text{N} : \text{A} \]

\[ \Gamma + (\text{A} : \text{A}, \text{M}(\text{N}) \in \text{P}/\text{A}) \quad \text{universal} \]

Notice: Very strict! And no hypothetical reasoning.

\[ \text{Suppose you also had the rules for } \text{N.} \quad \text{End } \text{N.} \quad \text{(44)} \]
Then you can define double & plus.

But

\[ \text{ax. nat. double (w) } \neq \text{ Veblen, not } \]

\[ \text{ax. nat. plus (n, n) } \]

even though double (w) = plus (m, m).

Similarly:

\[ \text{ax. nat. plus (0, 0) } = \text{ ax. nat. n } \]

\[ \text{ax. nat. plus (0, 0) } \neq \text{ ax. nat. n } \]

Equality, leading to dependency.

Last time: in formal type theory, we have
defeq equivalence

\[ \Gamma \vdash M =: A \]

Axiomatized as the least congruence containing
\( \beta \) rules. (Note: Methodologically intended
to be decidable (never efficiently so, as Hessen showed). A nice issue is: Shannon expansion.)

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

Motivation "Binarize logical relations"

\( F \vdash \Gamma \vdash M : A \), then \( \Gamma \vdash M \equiv A \), where

\( \Gamma \vdash M \equiv A \) mean \( \forall \gamma, \text{HT}_A (\gamma) \) implies \( \text{HT}_A (\gamma(10)) \).

Want at a min: if \( \Gamma \vdash M \equiv A : A \), then \( \Gamma \vdash M \equiv 1M \equiv 1A \equiv A \)
Also want a characterization of exact equality in terms of program behaviour.

**Notation:**

- \( M \in A \) means \( H_1 \cdot (M) \)
- \( \Gamma \Rightarrow M \in A \) means \( \forall \text{ } Y: \Gamma. \ H_1 \cdot (\text{ } Y \text{ } ) \) implies \( H_1 \cdot (\text{ } M \text{ } ) \)
- \( M \cdot \equiv \cdot N \in A \) means \( E_\cdot (\text{ } M \text{ }, \text{ } N \text{ } ) \)

Let's define \( M \cdot \equiv \cdot N \in A \) means \( M \equiv N \in A \).

\( \vdash \) will be shown to be a PER

\( \Gamma \Rightarrow M \equiv M' \in A \) means extensional / functional, maps (open terms) are functions.

**Explicitly,** if \( Y := Y' \in \Gamma \), then \( \text{ } Y \cdot (\text{ } M \text{ } ) \equiv \cdot \text{ } Y' \cdot (\text{ } M' \text{ } ) \cdot \in \cdot A \)

- e.g., \( x : A \Rightarrow M \in B \) defines a function that respects exact equality in \( x : A \)
- if \( P \equiv Q \in A \), then \( \text{ } P[x] \cdot M \equiv \cdot Q[x] \cdot M \in \equiv B \).

Let's go through all of the judgments again: 

- **Logical equality** \( M \equiv M' \in A \) iff either \( M, M' \cdot \equiv \cdot M \) or \( M, M' \not\equiv \cdot M' \not\equiv \cdot M \)
- \( M \equiv M' \not\equiv \cdot M \) and \( N \equiv N' \not\equiv \cdot N' \)
- \( M = M' \in A \) iff \( M \equiv M' \in A \), \( M, M' \cdot \equiv \cdot M' \equiv M \), and if \( M = M' \in A \); then \( \text{ } M'[x] \cdot M \equiv \cdot [\text{ } M'[x] \cdot M \equiv \cdot M' \equiv M' \equiv M \).

**Fact 1)** \( M = M' \in A \) implies \( M' = M \in A \).

**Fact 2)** \( M = M' \equiv M'' \equiv M'' \in A \) implies \( M = M' \equiv M'' \equiv M'' \in A \).

**Proof:** By induction on \( A \). Use determinism of reduction in \( \overline{\overline{\text{ } A \text{ } }} \).

Aside: parametricity = semantic equivalence for polymorphic types. 

\( M = M' \in A \) iff Girard's Method / Reynolds' Method

**Funny:** you can't show directly the transitivity of parametricity. The reason is, when you use Girard's method, you're no longer working with equivalence relations. You're working in heterogeneous relations, they're these simulation relations. You can't show the equivalence of candidate coreference types to client. If you understand this, you understand data abstraction, and really, you're data abstraction.

\[ \text{\textcopyright{} 1996} \]
Fact (head expansion):
if \( M \equiv N \in A \) and \(N \vdash M') \implies N \vdash M \) and \(N' \vdash M'\), then \(N \equiv N' \in A\).

Definitional equivalence suffices for exact equality.

Pf: By induction on the derivation.

\[\frac{\gamma \vdash M \in A}{\gamma \vdash M \equiv M \in A}\]

\[\frac{\gamma \vdash M' \in A \quad \gamma \vdash M \equiv M' \in A}{\gamma \vdash M \equiv M' \in A}\]

Lemma: If \(M \equiv M' \in A\), then \(\Gamma \vdash M \equiv M' \in A\).

Pf: Assume \(\gamma \vdash M \equiv M' \in A\). Because \(\gamma, \gamma'\) are closed,

\[\gamma \vdash M \equiv M' \in A\]

By assumption, \(\gamma \vdash M \equiv M' \in A\), so by sym., \(\gamma \vdash M \equiv M' \in A\).

\[\frac{\gamma, x \in A \vdash M \equiv M \in A \quad \gamma, x \in A \vdash M \equiv M \in A}{\gamma, x \in A \vdash (\lambda x : A. M) \equiv M \in A}\]

Suppose \(\gamma \equiv \gamma' \in \Gamma\). TS: \(\lambda x : A. M(\gamma x) \equiv M(\gamma) \in A\).

\[\frac{\gamma \vdash M = A \quad \gamma \vdash M' = A}{\gamma \vdash M \equiv M' = A}\]

Suppose \(\gamma \equiv \gamma' \in \Gamma\). TS: \(\lambda x : A. M(\gamma x) \equiv M(\gamma) \in A\).

\[\frac{\gamma \vdash M = A \quad \gamma \vdash M' = A}{\gamma \vdash M \equiv M' = A}\]

Pf: Use head exp.

Fact: Assume \(x \not\vdash \Gamma \Rightarrow M, M' \in A\). WTS when
\(x \not\vdash \Gamma \Rightarrow M = M' \in A\). Claim:

STF: 1) \([C0\{x\}] M = [C0\{x\}] M' \in A\]

2) if \(N \vdash M \equiv M' \in A\), then \([N/x] M \equiv \alpha(N) \vdash M \equiv M' \in A\)

Proof: \(\vdash \) is pretty sure the above is right. Not check it.
You might need to change 1) to

1) \( \text{if } N \neq 0 \text{ and } N' \neq 0, \text{ then} \)
\( \frac{N}{x} M = \frac{N'}{x} M \cdot E A \)

and update 2) in a similar manner.

Remark: We aren't using mathematical induction, but rather the inductive structure of the type \( \text{nat} \).

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

\[ [y/x]M = [y/x]M' \cdot E = \gamma \cdot \text{new} \left( \frac{\text{eq}(y)}{x} M = \frac{\text{eq}(y)}{x} M' \cdot E A \right) \]

functionally/generically in \( y \)

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

\[ \gamma \cdot \text{new} \left( \frac{\text{eq}(y)}{x} M = [y/x]M' \cdot E A \right) \Rightarrow \frac{\text{eq}(y)}{x} M = [y/x]M' \cdot E A \]

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

\[ \gamma \cdot \text{new} \left( \frac{\text{eq}(y)}{x} M, [y/x]M' \right) \Rightarrow \exists x : E A \left( \frac{\text{eq}(y)}{x} M \right) \]

Remark: \( E A \) is a type in terms in \( i \). This is where \( E A \) dependent type comes in.

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

1. Equational reasoning in formal and semantic type theory
   1.1. $M \equiv N \iff \exists \alpha . \alpha \in \text{Mat} \to \alpha(P(M,N))$
   1.2. $\Pi$ if $M \not\equiv N$, then $P(M,N)$
   1.3. if $M \not\equiv N$, then $\alpha(M) \not\equiv \alpha(N)$ and $\alpha(P(M,N))$, then $P(M,N)$

2. Broached topic of proof-as-types principle, leading towards deep type.

Modes of reasoning are very similar to the historic modes of reasoning used to define type constructivist.

[proofs-as-types aka proof-as-programs]

[unification of logic and types]

Two key players:

1. Brouwer - developed a theory of truth [Church]
2. Hilbert - a theory of formal proof [Russell]

Semantics vs. syntax,

truth vs. proof,

derived from the

axiomatization

of algorithms

What does it mean for $M \rightarrow A \rightarrow B$ to be
true? Brouwer: it 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 is true.

M : A is by definition meaningful.
M ⊨ A is by definition meaningful.

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

**Formal proof (in logic)**

1. **Hilbert-type system** (combinators)
2. **Gentzen-type system** (A-terms)

**Hilbert system** - emphasis on unconditional truth.

```
A ⊨ A  true
A ⊨ (B ⊩ A) true
(A ⊨ B ⊢ C) ⇒ (A ⊨ B) ⊢ (A ⊨ C)
```

**Gentzen-type system**

Based on entailment and hypothetical reasoning.

```
block  [type A true]
structure [B true]
A ⊨ B true
```

Entailment:

```
A ⊨ A true  B ⊨ B true
```

ניק קרלוזט

50
Proof simplification

You have a rule:

\[
\begin{align*}
A & \rightarrow B \rightarrow C \\
A & \rightarrow B & \rightarrow C & \rightarrow D \\
A & \rightarrow B & \rightarrow C & \rightarrow D & \rightarrow E \\
A & \rightarrow B & \rightarrow C & \rightarrow D & \rightarrow E & \rightarrow F
\end{align*}
\]

When are two proofs the same?

When are two algorithms the same?

Proof simplification is based on substitution.

Church: \(\lambda\)-calculus based on substitution.

Formal correspond bob \(\lambda\)-calc + pfs.

This should be renamed to Church-Sentey- correspondence.

This is a meaningless correspondence, and owes of historical relevance.

Formal correspondence:
1. Pre-equity logics correspond to typing systems for \(\lambda\)-terms.
2. w/ Sentey's input, induced a notion of \(\Pi\) equivalence
   some equiple fundamental.

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

\[
\begin{align*}
\Pi & + M1 \rightarrow A \\
\Pi & + M2 \rightarrow N \rightarrow A
\end{align*}
\]

In equivalent justifications.
Until Gentzen, there was no notion of equality and the concept doesn't even line up properly. Plus, what properties does "not" convey to? The "Curry-Howard isomorphism" is meaningless, correspondence between a meaningless theory and something of interest. Only def. terminals note.

**Brouwerian conception**

Theory of truth, not formal proof.

Based on proof which is self-realis

A human dialog.

**Prop. Judgment** $M \vdash A \quad M \vdash \text{proof of } A$

Evidence for constructively understandable, truth of $A$.

$M = N \vdash A$

**Semantic Conv**

<table>
<thead>
<tr>
<th>$T$</th>
<th>$0$</th>
<th>$\text{not } A$ (partial)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \land B$</td>
<td>$A \land B$</td>
<td></td>
</tr>
<tr>
<td>$A \lor B$</td>
<td>$A \lor B$</td>
<td></td>
</tr>
<tr>
<td>$A \rightarrow B$</td>
<td>$A \rightarrow B$</td>
<td></td>
</tr>
<tr>
<td>$\neg A = A \lor L$</td>
<td>$A \land 0$ (A const)</td>
<td></td>
</tr>
</tbody>
</table>
There are many types that have nothing to do with logic and $TT$ comes first.

Eg. $\neg A$ is partial force and hence it need not terminate. So no carry to truth.

Unfamiliar issues:

1. $A \lor A$ (for any $A$)
   $A \lor (A \rightarrow 0)$

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

2 things that were missed until now:

1. Weaker conjunction: $\neg \neg (A \lor A)$
2. $\neg (\neg A \lor \neg A)$

You can recover everything from classical $\land$, it just depends which version of $\lor$ you actually mean.

2. You can postulate an oracle $O_0 \in A_0 \rightarrow (A_0 \rightarrow 0)$

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

Type! It is no more restrictive than $\mathcal{C}$.

Points: too much is made of the formal concept. The real one is the semantic concept.
Formal Dependent Type Theory

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

This is a formal type theory - inductively defined.

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

In particular: \( I_A(M,N) \), where \( M,N \in A \) \( \forall \alpha \in \text{Vec}(M) \) where \( M : \text{nat} \)

Today: "core" ITT with \( E, \Pi, I_A \).

Syntactic judgments look like those we had before.

\[
\begin{align*}
\Gamma & \vdash M : A \\
\Gamma & \vdash M = M' : A \\
\Gamma & \vdash \text{ctd} \\
\Gamma & \vdash A = A' \\
\Gamma & \vdash \alpha \in \Gamma 
\end{align*}
\]

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

Structural Properties (axiomatically)

(R) The key points is that "\( \alpha \)" can appear in the type in \( \Gamma, \Gamma' \). The contexts are ordered.

Remark that \( A \) is a type in \( \Gamma \) is the \( \text{ctd} \) \( \Gamma \), and it remains a type when we weaken with \( \Gamma' \).

\[
\begin{align*}
\Gamma & \vdash A \text{ type} \\
\Gamma & \vdash B \text{ type} \\
\Pi & \vdash A \text{ type} \quad \Gamma_B & \text{ type} \\
\Pi & \vdash A \text{ type} \quad \Gamma & \vdash B \text{ type} \\
\end{align*}
\]

"B type" could be replaced by any atomic judgment. \( \Gamma \)
"order doesn't matter (P) unless it does\" E is R, S, T and compatible constructs

\[ \Gamma \vdash a : A \quad \delta \vdash A, \Gamma' \vdash J \]

\[ \Gamma, z : A, [z, z] \delta \vdash A \Gamma' \vdash [z, z] \delta \vdash J \]

\[ \Gamma \vdash M : A \quad \Gamma, a : A \Gamma' \vdash J \]

\[ \Gamma \vdash [M / a] \Gamma' \vdash [M / a] J \]  \hspace{1cm} \text{(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 \( \lambda x. y. x \).)

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

\[ \text{Substitution} \]

\[ \Gamma, M \vdash A \quad \Gamma / M \vdash A = A' \]

\[ \Gamma / M \vdash \text{M} = M' \]

What should \( A = A' \) be? It could be empty. But choosing a "good" \( A \) doesn’t affect everything. It makes no difference in the strongest typed setting, but in the dependently typed setting it really matters and is induced by term equivalence.
Booleans

Surprise: hard to capture.

Γ : Bool
Γ + tt : Bool
Γ + ff : Bool

Want to write:
Γ : Bool
Γ + P : A
Γ + Q : A
Γ + if (M ; P ; Q) : A

\{ Γ + if (tt ; P ; Q) = P : A
   Γ + if (ff ; P ; Q) = Q \}

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
   \[ \text{for } \text{all } \] or \text{of decidability,} \]
   if (M ; P ; Q)
   doesn't have a clean type: you could read
   Γ + P : A
   and then what do you do?
   So we add a type annotation
   Γ P (M ; P ; Q).

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

Suppose, indeed, that you have
Then \( \Gamma \) and \( A \) and \( Q \) could depend on \( A \). So you want to generalize the type of the condition and propagate upwards.

\[
\Gamma \vdash A \quad \Gamma \vdash Q : [\text{ff}/a]A
\]

\[
\Gamma \vdash \text{if} \ a. \ A (\text{M};P;Q) : [\text{M}/a]A
\]

What are meaningful examples of \( a \cdot A \)? Well, you might want to write something of the form

\[
\text{if} \ a. \ ? (\text{M};17;"17") : \text{if} \ CM, \text{Nat, Std}
\]

???

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

This problem is not restricted to \( a \cdot \text{Bools} \)

but rather all positive types as a mapping our profoundly

3) "Large Elimi": mapping into type.

4) The issue is the Shannon exp is:

It's the universal property for the Bools.

But you can't get this in formal setting.
There's an issue as to whether you treat $\Sigma$ as pos or neg. Well, treat $\Sigma$ negatively.

**Before**

\[ \Gamma \vdash A \text{ type } \quad \Gamma \vdash B \text{ type } \quad \Gamma, a : A \times B \text{ type} \]

**Now**

\[ \Gamma \vdash A \text{ type } \quad \Gamma, a : A + B \text{ type } \]

\[ \Gamma \vdash a : A \rightarrow B \text{ type } \quad (\Pi x : A, B) \]

\[ \quad a : A \times B \text{ type } \quad (\Sigma x : A, B) \]


\[ \Gamma \vdash \lambda a : A. \mu : a : A \rightarrow B \]

\[ \vdash \lambda a : A. \mu : a : A \rightarrow B \]

\[ \vdash M : a : A \rightarrow B \quad \vdash N : a \]

\[ \vdash M(N) : (\text{N/}a) B \]

\[ \vdash (\lambda a : A. M) N = (\text{N/}a) M : (\text{N/}a) B \]

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

\[ \lambda a : A. B \]

But lazy, so...
\[ \Gamma \vdash A \quad \Gamma, a : A \vdash \text{type} \]
\[ \Gamma, a : A \vdash B \quad \Gamma, a : A \vdash \text{type} \]
\[ \Gamma \vdash (a, b) : a \times b \]

**Negative formulations of \( \Sigma \):**

\[ \Gamma \vdash A \quad \Gamma, a : A \vdash b : B \]
\[ \Gamma, a : A \vdash b : B \]
\[ \Gamma \vdash \text{fst}(a, b) : A \]
\[ \Gamma \vdash \text{snd}(a, b) : B \]

\[ \Gamma \vdash \text{fst}(a, b) : A \]
\[ \Gamma \vdash \text{snd}(a, b) : B \]

**Identity Types**

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

Idea: internalize "equality".

1. Defn equivalence is defined for typed terms, but in not typed dependent.

2. "RWH forget DK"

3. "Judgmental concepts precede type concepts".
Just like you have the judgmental motion of map

\[ \Gamma, a : A + M : B \]

which you internalize as

\[ \Gamma, \Gamma a : A, M : a : A \rightarrow B, \]

you want to do the same for equality.

\[ \Gamma \vdash A \text{ type} \quad \Gamma \vdash M, N : A \]

\[ \Gamma \vdash \text{Id}_A (M, N) \text{ type} \]

What is \( \text{Id}_A (M, N) \)? It's the least reflexive relation. This means when you think it is though and has been the cause of much consternation.

\[ \Gamma \vdash A \text{ type} \quad \Gamma \vdash M : A \]

\[ \Gamma \vdash \text{refl}_A (M) : \text{Id}_A (M, M) \] (I)

Family of types

\( \text{Id}_A (M, N) \) is a family of types over \( A \times A \).

For particular choices of \( M, N \), it may not be populated.

This type is a subtype argument.

We use this terminology instead of saying "0 or 1 initially because we don't necessarily know which".
The leastness is given by a mapping and property. For the induction
\[
\Gamma, \ a : A, \ b : A, \ c : \text{Id}_A(a,b) \rightarrow C \text{ type} \quad \text{"the motive"}
\]
\[
\Gamma, \ a : A \vdash Q : \text{Id}_A(a,b) / \text{a b c C}
\]
\[
\Gamma \vdash \text{Id}_A(a.b)(P) : \text{C} / \text{a b c C}
\]

\[
\text{Fact}
\]
1) Symmetric:

\[
\text{Sym}_{A,B,C}(a) : \text{a} \cdot \text{Id}_A(M,N) + \text{sym}_{(a)} \cdot \text{Id}_A(N,M)
\]

2) Transitive:

\[
a : \text{Id}_A(M.N), \ b : \text{Id}_A(N,P) \rightarrow \frac{a}{b} : \text{Id}_A(M,P)
\]

\[
\text{trans}_{A,M,N,P}(a,b)
\]

Exercise: Find sym + trans.

Trans: Use Yoneda.
Ex: Show that the following

\[ \text{id}_A = \text{sym}(\text{refl}(A)), \text{id}_{\text{refl}(A)} = \text{refl}(A). \]

\[ \text{id}_A (a, b) \in \text{refl}(A) \]

\[ \text{id}_A (a, b) \in \text{refl}(A) \]

Gropaid laws.

Notation is cumbersome, so we write

\[ =_A \] for the type \[ \text{id}_A (-,-) \]

\[ =_A \] is an equivalence relation.

1. Reflexivity

\[ \Gamma, a : A, \Gamma : \text{refl}_A(a) \]

2. Symmetry:

\[ \Gamma, a : A, b : A, \Gamma : a =_A b \text{ true} \]

Witness:

\[ \text{sym}(a,b_\Gamma)(\Gamma \times_A b) : b =_A a \]

often abbreviated as \[ \Gamma \times_A b \]

\[ \Gamma, a, b, \Gamma : b =_A a \]

\[ \Gamma, a, b_\Gamma, \Gamma : b =_A a \]

Note: \[ \text{sym}(a)(a) \text{refl}(a) = \text{refl}_A(a) \]

is a consequence of the exact term and

\[ \text{define symmetry.} \]
3. Transitivity:

\[ a, b, c : A, \ p : a = b, \ q : b = c, \ a = c \]

Witness: \( \text{trans}(a, b, c, p, q) \)

Notation: \( (g, p) q \)

Auxiliary:

\[ a, b : A, \ p : a = b, \ (c : A \rightarrow (b = c \rightarrow a = c)) \]

(An isomorphism version of Yoneda)

\[ \mathcal{J}(a, b, c : A \rightarrow (b = c \rightarrow a = c)) (a, \lambda c : A \rightarrow \text{id}_{a = c}; p) \]

\[ \vdash (b = c \rightarrow a = c) \]

Then apply this to \( c \) and \( q \).

Note: \( \text{trans}(a, b, c, p, q) \neq \text{refl}_A(b) q \)

\( \text{trans}(a, b, c, p, q) \neq p \)

Summary:

\[ \text{refl}_A(M) \text{ aka } \text{id}(M) \]

\[ \vdash \text{symmetry} \]

\[ \vdash \text{trans} \]

\[ \text{id}(M) \dashv \vdash \text{id}(M) \]

\[ \vdash \text{id}(M) \cdot q = q \]

Tantalizing thought: This looks like group structure.

1. Respect for identity, aka transpar, aka Leibniz's principle of indiscernibility of identities.
\[ \Gamma \vdash p : M = N \]
\[ \Gamma, a : A \vdash B \quad \text{by prop} \]
\[ \Gamma, a \vdash [a.B](p) : B[M] \rightarrow B[N] \]

Definable from \( T \) such that
\[ \Gamma \vdash [a.B](\text{refl}_A(M))(R) = R \]

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

\[ \Gamma \vdash [a.B](p) \]
\[ \Gamma \vdash [a.B](p) \leq \begin{array}{c}
\text{B [a] } \rightarrow \text{B [a]}
\end{array} \]
\[ \Gamma \vdash [a.B](p) \leq \begin{array}{c}
\text{refl}_A(M) \rightarrow \text{refl}_A(M)
\end{array} \]

Because of the way you wrote the code, yet
\[ \Gamma \vdash [a.B](p) \leq \begin{array}{c}
\text{refl}_A(M) \leq \text{refl}_A(M)
\end{array} \]

but this is purely accidental

\[ \vdash \text{"Id is a notion of equality"} \]

Or is it?

Consider the following:
\[ \text{\( b : \text{Bool} \vdash \text{if}\ b \ (b , \text{true} , \text{false}) = b = b \text{-type} \) } \]

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

But it is not the case that
\[ \vdash \text{\( \lambda b : \text{Bool} \vdash \text{if}\ b \ (b , \text{true} , \text{false}) = b = \text{~true} \) } \]

\[ M =_{\text{Ann}} N \text{ true iff } M = N : A. \]

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
it and your notion of equality, then you are screwed.

Proof is by induction on \( b \).

\[
\begin{align*}
\text{Hypothesis: } & \quad \text{Book} \left( a, b, f f \right) = \text{Book} a \text{.} \\
\text{Conclusion: } & \quad \text{Book} \left( b, b, f f \right) = \text{Book} b \text{.}
\end{align*}
\]

Terminology: You do not have function

\text{extensionality}

intensional \neq \text{not} \text{ extensional}

What to do?

0) be happy: It is intensional definition equality.

1) But: I want Book \to Book to be a type

of functions (Boy, you! The secret to
happiness is not to want too much).
a) Go straight to setoid hell.

Setoid = type + equiv reln. 
Ways = Herbrand equivalence.

Then take:

\[(\text{Bool} \rightarrow \text{Bool})/\sim\]

where

\[\sim(F,G) \iff \forall x : \text{Bool}. F(x) \approx_{\text{Bool}} G(x).\]

b) Check: ETT.

Postulate equality reflection

\[\Gamma \vdash P : \text{Id}_A(M,N) \quad (ER)\]

\[\Gamma \vdash M = N. \Delta\]

and uniqueness of

\[\Gamma \vdash P,A : \text{Id}_A(M,N) \quad (UIP)\]

identity proofs.

\[\Gamma \vdash P = A : \text{Id}_A(M,N)\]

Perfectly valid on closed terms!

Except type checking ≠ proof checking,

which blows the entire hopes of

\[\text{ETT out of the water}\]

c) wisc: add new elements of \(\text{Id}_A(\_\_\_\_\_\_\_\_\_\_\_\_\_\_)\).

i) \(\text{FUnExt}_{A,B} : \text{TT}_{F,G} : A \rightarrow B. (\forall a : A. F(a) \approx B(a)) \rightarrow \text{Id}_{B \equiv B}\)

Adding this constant/expansion breaks M-L's

Law.

What else is wrong? What's the context? What is

\[\text{OTT} \quad \text{JT} \rightarrow \text{JT} \quad \text{FUnExt}_{A,B} (F,G,P) \equiv ???\]

\[\rightarrow \text{ad-hoc answer to fit}\]
ii) Univalence Axiom (Voevodsky) (later).

"Bury i" some new term UA

Equiv(A, B, E) → Id_{A, B}

Isomorphism: unweirc

You still have the exact same problem.
What would a comlp interp of ITT be?

Informally, want to run closed terms as free vars.

1. Frames – types are irrelevant at run-time
   Extraction – recovering a “new 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 Hypothesis" — FJLR

Recall

1. We gave a semantics in terms of Computation for \( \mathbb{N} \), which did not have type

\( \mathbb{N} \)

- If \( \mathbb{N} \rightarrow \mathbb{A} \), then \( \mathbb{N} \rightarrow \mathbb{1} \mathbb{M} \mathbb{1} \mathbb{A} \)

- Tait's, Girard's, and Reynolds' methods

NB: Start with programs. Extraction is fundamental.

How do we generalize this to dependent types?

Crucial idea: Types are programs. Specifications. They run and yield programs.
E.g. if \((H, 17, "17") \in \text{if}(M, \text{Nat}, \text{Str})\).

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 meaning as follows:

\[ \Gamma \vdash A \text{ type implies } \Gamma \Rightarrow A \text{ type} \]

where \(\Gamma \Rightarrow A \text{ type}\) means at least that \(A\) is well-behaved as a program. By abuse of notation,

\[ \text{HT}_{\Gamma} (A) \]

\[ \Gamma \vdash A = B \text{ type implies } \Gamma \Rightarrow A = B \text{ type} \]

"EQ\text{_{\Gamma}} (A, B)"

**Computational Defn Type Theory**

Given an op sem for closed erased terms

\[ A, M : V \]

\[ A, M \rightarrow V \]

Define the \text{iff} semantic judgments:

\[ A \text{ type} \]

\[ "\text{I know that } A \text{ is a type}" \]

\[ A = B \]

\[ "\text{I know that } A \text{ and } A' \text{ are equal type}" \]

\[ M : A \]

\[ "\text{Given that I know } A \text{ type, I know that } M \text{ exhibits the behaviour specified by } A/ \text{satisfies } A" \]

\[ M = M' : A \]

\[ "\ldots \text{ } M \text{ and } M' \text{ equisatisfy } A" \]

A type means \(A : V\) and \(V\) (names a specification of behaviour)

The literature says: "\(V\) is a canonical type", where "canonical type" should be read as a noun "canonical type", rather than an adjective. Adjective, i.e., a special kind sort of type."
eg) Book is a specification (see below)

\[ \text{Book} \rightarrow \text{Book} \]
\[ \text{if(M, Not, Shy)} \text{ evaluates to a spec.} \]

\[ A = A' \text{ means } A \sqcap V, A' \sqcap V', V \text{ and } V' \text{ are equivalent/inal spec.} \]

**MCA** where \( A \sqcap V \) spec

- \( M \sqcap W \) and \( W \) satisfies \( V \)
  - \( W \) is a Canonical element of \( V \)

eg) \( \text{the sets Book} \)

\[ \text{if } \text{sats Book, } \text{then Book is inductive} \]

**M = M' e A** where \( A \sqcap V \) spec

- \( M \sqcap W, M' \sqcap W' \), \( W \) and \( W' \) satisfiy \( V \)

(Subsumes def'n equivalence)

\[ \text{Eq.: } \text{id} = \text{id} \in \text{Non} \rightarrow \text{Non} \]
\[ \text{id} = \text{id} \in \text{E} \rightarrow \text{E} \]

All of this can be consolidated as

- \( A = A' \)
- \( M = M' \in A \) equal specs

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

(PERs)

- \( V \) and \( V' \) are equal specs
- \( V \) and \( V' \) eq. sat \( W \)

\[ \text{Eq.: } \langle M, N \rangle \in A \times B \text{ (value)} \]
\[ \text{iff } M \in A \text{ and } N \in B \text{ (non-value)} \]

There are categorical judgments (stated w/o any conditions), need to be confused w/ categorical judgments.
We need hypothetical judgments

(in M-6: hypothetical → general)

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

1) \( \alpha \Rightarrow I \) means \( I \).
2) Given \( \alpha : A \Rightarrow B \) type means
(A) (first, insufficient)
\text{if } M \in A \text{ then } [M/a]B \text{ type}

Want: need more, because types are defined by PERs, and you need to ensure this is expected. \( M : A \) does not pick one a canonical rep for the equiv class, it's just a piece of code.
So you need:

\text{if } M \equiv M' : A \text{ then } [M/a]B \equiv [M'/a]B \text{ type}

And when you really mean, seeing that you're working with PERs, \( \alpha : A \Rightarrow B \) type means

(B) \( \alpha : A \Rightarrow N : N' \in B \) (given that \( \alpha : A \Rightarrow B \equiv B \))

\text{if } M \equiv M' : A \text{ then }
[M/a]N \equiv [M'/a]N' \in (M/a)B \equiv C(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}) \text{ctn}

a_1 : A_1, \ldots, a_n : A_n \quad a \equiv A_1 \quad M^* \equiv M', eA

"iterated functionality" (CMCP)

\[ \Rightarrow \quad \text{Check that the formal structural properties are validated as hypothetical} \]

\[ P, a : A, \Gamma \Rightarrow N eB \quad \Gamma \Rightarrow M eA \]

\[ \Gamma \left[ M / a \right] \Gamma \Rightarrow \left[ M / a \right] N e \left[ M / a \right] B. \]

You'll discover that all of the properties of hypothetical judgments are true! We'll have more than just entailment, you've given a semantics of variables.

ey)

**Bool as a type**

1. **Bool** and **Bool** are equal types.
2. \( \text{tt} \) and \( \text{tt} \) equally satisfy \( \text{Bool} \).

   or \( \text{ff} \) nothing else.

\[ \forall v, v' \quad e : \text{set} \quad \text{Bool} \quad / /
\]

either \( v = v' = \text{tt} \)

or \( v = v' = \text{ff} \). \( \) (Sytntetically equal)

**Fact:** This is not a definition! \( \triangleleft \)

It is a fact:

If \( b : \text{Bool} \Rightarrow A = A \) \( \) (note=)

\[ M \equiv M', e \text{Bool} \]

\[ N \equiv N', e [\text{tt/1b}] A \]

\[ P \equiv P', e [\text{ff/4b}] A, \]

then \( (M, N, P) \equiv (M', N', P') \) \( e [\text{H/6}] A \).

We will see a worst next time.
Recall

hypothesical (general) judgments define what is a mapping functionally,

\[ x_1: A_1, \ldots, x_n: A_n \vdash A \]
\[ M \equiv M' \in A \]

\[ n = 0: \text{ categorical} \]
\[ n > 1: \text{ consider} \]
\[ x_1: A_1, \ldots, x_n: A_n \text{ type} \]
\[ x_1: A_1, \ldots, x_n: A_n \vdash \left\{ \begin{array}{l} A \vdash A' \\ M \equiv M' \in A \end{array} \right. \]

Consider all pairs of instances,

\[ M_i \vdash M_i' \in A_i \]
\[ M_2 \vdash M_2' \in \left[ M_1/k_2 \right] A_2 \]

... then
\[ \left[ M_i/k_i \right]_{i=1}^n A \vdash \left[ M_i/k_i \right]_{i=1}^{n+1} A' \]
\[ \quad \quad \quad \quad \quad M \equiv M' \in \left[ M_i/k_i \right]_{i=1}^n A. \]

Check for categorical, and then hyp judgements

1. Symmetry \{ Check \}
2. Transitivity \{ Check \}

Use the "PRR trick" if \[ M \equiv M' \in A \] then \[ M' \equiv M \in A \]

Check structural properties of hypothetical, come to logical entailments:

1) \[ \Gamma, x: A, \Gamma' \vdash x: A \text{ (reflexivity)} \]
2) if \[ \Gamma \vdash J \] and \[ \Gamma + A \text{ type} \], then \[ \Gamma, \Gamma, A + J \text{ (weakening)} \]
3) if \[ \Gamma, x: A, \Gamma' \vdash J \] and \[ \Gamma, M \equiv M' \equiv A \], then

\[ \Gamma \vdash [M/x] J \equiv [M'/x] J. \]

Once the computational framework is setup, we can see how logic is embedded.

1. Evaluation
   \[ [x: A \text{ true}, \Gamma' \vdash A \text{ true}] \]

2. plus compose introduce/impose concept
of (exact) equality of proofs.

2. Dependent on the prop.

This is the semantic prop-as-types principle.

Next

Populate the theory w/ types.

1. `\text{Bool} \equiv \text{Bool}`.

Define: Least type containing the \( \text{ff} \) properties of conditionals are derived from this definition.

2. \( \Pi x : A. B \), aka \( x : A \rightarrow B \)

\( \Sigma x : A. B \), aka \( z : A \times B \).

Key: the claim forms are theorems/facts.

Function equality is extensional (wrt \( \equiv \))

\[ \lambda(x. M) = \lambda(x. M') \equiv x : A \rightarrow B \]

\[ \text{equality of behaviour} \]

\[ \text{if} \quad x : A \rightarrow B \]

\[ \text{closed values} \]

\[ \text{Respect for equality} \]

Equality Types

Internalize judgmental equality

\[
\begin{align*}
E_{B_A} (M, N) &\equiv E_{B_{A'}} (M', N') \\
\text{iff} &
\begin{cases}
A \equiv A', \ M \equiv M' \in A \\
N \equiv N' \in A
\end{cases}
\end{align*}
\]

\[ \text{refl} \]

\[ \ast \equiv \ast \in E_{B_A} (M, N) \quad \text{iff} \quad M \equiv \text{NEA} \]

Could write "refl" instead of "\( \ast \)", and \[ \text{refl}(M) \equiv \text{refl} \]
Note: refl $\in \text{Eq}(A, A, \lambda a. a, \lambda a. a a)$

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

1) If $\Gamma \vdash M : A$ then $\Gamma \Rightarrow IM \vdash A$

2) If $\Gamma \vdash M = M' : A$, then $\Gamma \Rightarrow IM \equiv IM' \vdash A$

3) If $\Gamma \vdash A$ type, then $\Gamma \Rightarrow \lambda A \vdash A$

4) If $\Gamma \vdash A \equiv A'$, then $\Gamma \Rightarrow \lambda A \vdash A'$

---

Crucial Part

\[ \Gamma \vdash \lambda A, B, C : \text{Id}_A(M, N) \uparrow \Delta \]

\[ \Gamma \vdash \lambda A, B, C : \text{Id}_A(M, N) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]

\[ \Gamma \vdash \lambda A, B, C : \text{Eq}(A, B) \]

\[ \Rightarrow \]
Why have we defined anything at all?

Need to know:
- \( \alpha : A \rightarrow B \)
  - \( A \) type, \( \alpha : A \rightarrow B \) type
- \( \alpha : A \times B \)
- \( \text{Eq}_A(M, N) \)

Principles of definition of these?

- Predicativity: structural induction
- You need to know what came prior
- "local defns" such as \( \text{Bool} \).

What exactly justifies the construction of a type system?

Recall: With simple types it's easy.
1. Define types \( A := b|A, \rightarrow A \rightarrow A \).
2. Define predicates by said on the structure of \( HT_A \rightarrow A \rightarrow HT_A \).

With dependency, it's thicker because types are tied up with elements.
- \( \text{Eq}_A(M, N) \) type when \( A \) type
- \( P \in \text{Eq}_A(M, N) \) when \( M \equiv N \in A \).

With inductive definitions, another concern arises.

Recall \( \text{Bool} \) values are the least \( \text{PER} \) over \( \text{tt}, \text{ff} \) s.t.

\[
\begin{align*}
\text{tt} & \equiv \text{tt} \in \text{Bool} \quad \{ \text{define values} \\
\text{ff} & \equiv \text{tt} \in \text{Bool} \\
\end{align*}
\]

Then \( \text{Bool} \) computations are

\[
\begin{align*}
\text{Nat} & \equiv \text{Nat} \in \text{Bool} \quad \text{iff} \quad \text{M} \equiv \text{N} \in \text{Nat} \quad \text{and} \quad \text{V} \equiv \text{W} \in \text{Nat} \quad \text{(Then) \, \text{Bool} \, \text{computations are}} \\
\text{Nat} & \equiv \text{Nat} \in \text{inductively defined as follows} \\
\text{zero} & \equiv \text{zero} \in \text{Nat} \\
\text{succ}(M) & \equiv \text{succ}(M) \in \text{Nat} \quad \text{when} \quad \text{M} \equiv \text{N} \in \text{Nat} \\
\text{M} \equiv \text{M} \in \text{Nat} & \equiv \text{iff} \quad \text{M} \equiv \text{M} \in \text{Nat} \quad \text{V} \equiv \text{V} \in \text{Nat} \quad \text{iff} \quad \text{M} \equiv \text{N} \in \text{Nat} \\ \\
\end{align*}
\]
Today we'll go after a more careful construction (E. Allen '87).

1. This should give us clarity going forward.
2. We can "calibrate the metatheory":

   Where does the construction take place? Deal with the infinite regress of justifications for the semantic model.

   "Ordinal analysis" = calibrate using ordinals.

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

We have fixed points for monotone operators, by Tarski's Theorem:

Let $L$ be a complete lattice:
1. a pre-order $\leq$ on $L$,
2. all subsets have meets (g.l.b.) and joins (l.u.b.):
   - if $x \leq L$, then $\bigwedge x \in L$ and $\bigvee x \in L$.

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

Then Tarski's Theorem says:

An order-preserving function (monotone) $f : L \to L$ (L-complete lattice)
has a complete lattice of fixed points. In particular, $f$ has a least fixed point given by

$$\bigwedge \{ x \in L \mid f(x) \leq x \}.$$

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

When $f(x) \leq x$, we say "$x$ is closed under $f$", or "$x$ is $f$-closed".
We'll stratify our construction to leverage 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.

1. Candidate type system
   \[ T(A, B, \phi) \]
   \[ \phi(V, W) \]
   A, B — closed values
   V, W — values, "value rels in" e.g.,
   \[ T(Bold, Bold, \beta) \]
   along with "equal terms in",
   \[ \beta(\alpha, \lambda), \beta(\beta), \text{ nothing else.} \]

2. A real type system is a well-behaved candidate in the following sense:
   a) \( T \) is functional:
      \[ \text{if } T(A, B, \phi) \text{ and } T(A, B, \phi') \]
      \[ \text{then } \phi = \phi' \]
   b) \( \text{PER-valued: if } T(A, B, \phi) \text{, then } \phi \text{ is a value - PER (symm + trans)} \)
   c) type equality is a PER:
      \[ \text{if } T(A, B, \phi) \text{ then } T(B, A, \phi) \]
      \[ \text{if } T(A, B, \phi) \text{ and } T(B, C, \phi) \text{, then } T(A, C, \phi) \]
   \[ \Delta \phi \]
   Remark: \( \phi \) forms a complete lattice under inclusion

3. Define a monotone operator on candidate type systems:
   \[ \exists(T) \equiv \text{BOOL}(T) \cup \text{EQ}(T) \cup \text{PI}(T) \cup \text{SIGMA}(T) \]
Show monotonicity.

\[ T_0 = \mu \exists^F \text{ least fixed point} \]

is the type system defined by \( D \).

Show \( T_0 \) is a real type system.

4. Define truth relative to a real type system

\[
\begin{align*}
T = A \equiv B & \iff T^\prec (A, B, \bot) \\
T = M \equiv N \equiv A & \iff T^\prec (M, N) \text{ when } T^\prec (A, A, \varphi)
\end{align*}
\]

where the liftings are defined as follows:

- if \( \varphi \) is a value relation, then \( T^\prec (M, N) \iff M \uparrow V, N \uparrow W, T(V, W) \)
- if \( \varphi \) is a cand. type system, then \( T^\prec (A, B, \varphi) \iff A \uparrow V, B \uparrow W, T(V, W, \varphi) \)

5. Show that typical rules are true for \( T_0 \), the initial type system.

e.g. \( \frac{\forall \alpha}{T_0 \equiv M \in A \rightarrow B} \)

\[ \frac{\forall \alpha}{T_0 \equiv M \in A} \]

\[ \frac{\forall \alpha}{T_0 \equiv M(N) \in A} \]

Then \( \Rightarrow, \Rightarrow, (T_0, \Sigma) \) 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 \( \exists^F (T) \) explicit:

\[
\begin{align*}
\text{BOOL}(\bot)(\text{Bool}, \text{Bool}, \bot) & \text{ as before} \\
\text{EQ}(T)(A, B,)
\end{align*}
\]
**EQ(T) (A₀, A₀', E) iff (def'')**
1. \( A₀ = E_\lambda^A (M, N) \) \( \text{val} \)
2. \( A₀' = E_\lambda^A (M', N') \) \( \text{val} \)
3. \( \exists \varphi \text{ s.t. } T_\varphi (A, A', \varphi) \)
4. \( \varphi^* (M, M') \) and \( \varphi^* (N, N') \)
5. \( E \text{ (refl, refl) iff } \varphi^* (M, N) \iff \varphi^* (M', N') \)

**PI(T) (A₀, A₀', \varphi) iff (def'')**
1. \( A₀ = x : A \rightarrow B \xrightarrow{\alpha} A₀' = x : A \rightarrow B \xrightarrow{\alpha} \)
2. \( T_\varphi (A, A', x) \) \( (\exists \alpha) \)
   
   \[ \text{if } \alpha^* (M, M') \]
3. \( \exists \psi \text{ s.t. } (\text{Type x Type} \rightarrow \text{ValRel}) \rightarrow \text{K} \)
   
   \[ \text{for all } M, M', \text{ if } \alpha^* (M, M'), \text{ then } \psi (\alpha^* (M, M')) \]

4. \( \varphi (M, N, N') \) iff (def'')
   a.) \( M = x \alpha, N, M' = x \alpha, N' \)
   b.) \( \text{if } \alpha^* (P, P'), \text{ then } \psi (P, P') \supset (\alpha \alpha, N, \psi (P, P')) \)

**SIGMA(T) is analogous.**

---

Check: There are all monotonous.

The girl/core idea is that "A and all [M/x]B for HGA are prior to x:A x B". The [M/x]B are not substructures of x:A x B and 2:A x B. But they are prior and are never lost when you extend the type system by applying H.

If you try to formalize this, you need to be able to 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
   
   \[ \text{point} / \text{Acc(T)} (A, B, x) \]
2. Cannot "revise" the definition of any prior type by any monotone constructor. Consider the (bad) \( X \):

\[
X(t)(\text{Nat}, \text{Nat}, \eta')
\]

where \( t(\text{Nat}, \text{Nat}, \eta) \) and \( \eta' \neq \eta \).

Because previous levels depend on the definition of \( \text{Nat} \), and by changing \( \text{Nat} \), you may invalidate them.

---

2018-03-20

1.5 weeks ago, we saw pre-type systems. Recall we never defined an \( F(t) \) where

\[
F(t) \text{ is defined by a condition:} \\
\text{if } F(t) \text{ then } F(t)'' \rightarrow \text{you lose monotonicity.}
\]

Only ever of the form "if \( t \) then \( F(t)'' \)"

We defined 

\[
\text{To} = \text{iff } \Phi
\]

(Renamed \( \Phi \) to \( \Sigma \) today).

\( \text{To} \) is a pre-type system by construction.

1) functionality:

TS: if \( \text{To}(A, B, \Phi) \) then \( \text{if} \) \( \text{To}(A, B, \Phi') \)

then \( \Phi = \Phi' \) (\( \Phi(H, M, N) \) iff \( \Phi'(H, M, N) \))

(Note: A, B, M, N are values. We have definition of \( \phi, \phi' \).)

[Handwritten notes continue]
\( T(A, B, \varphi) \equiv \text{if } T_0(A, B, \varphi') \text{ then } \varphi = \varphi' \).

\( \text{S5: } T(2) \subseteq T \) "a pre-fixed point"

\( \vdash T_0 \subseteq T \). To show this, it is sufficient to consider each clause of \( T \).

Today: "Gödelian wheel" renamed to "Gödelian spiral".

Let's revisit formal type theory - \( \text{ITT} \):

Inductively defined.

We showed that if \( \Gamma + M : A \), then \( \left[ \Gamma \vdash \left[ M/\alpha \right] A \upharpoonright \text{ "measure" } \right] \).

Called out that \( \text{Id}_A(M, N) \) is not an adequate notion of equality, though it is a reflexive equivalence relation and has transport, it provides indiscernibility of identicals. But the empty relation also provides this. But type theory doesn't have a notion of function extensionality. The problem is that \( \text{Id}_A \) is defined uniformly in \( A \).

Indiscernibility of identicals:

Idea: families respect identity predicates.

\[ \{ \text{iff } B(M) \text{ true and } \text{Id}_A(M, N), \} \]

\[ \vdash B(N) \].

\( \Gamma, \alpha : A \vdash B \text{ Type } \)

\( \Gamma, \alpha : \text{Id}_A(M, N) \)

\( \Gamma, \alpha : [M/\alpha] A \vdash B \) proof of \([M/\alpha] B \)

\( \Gamma \vdash T(\ldots, P(Q)) : [N/\alpha] B \)

\( \text{Transport} \)
\[ \text{tr} \{a, B\} (\text{refl}_A(M))(Q) \equiv Q : [M/a]B \]

\(\text{β-rule / simplification}\)

The trouble arises because as the identity, it never does anything.

The idea is that Pis 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 \(I_{\text{tr}}\), it's not the family that knows this: it's built in to the system generically. And by \(M, a\), \(\text{Ann}\) the only identification is refl. So the whole setup breaks.

\text{WTS}

\[ \text{one} \]

\[ \text{two} \]

\[ \text{Id}_{\text{tr}}(\text{zero, sum} \text{ (zero)}) \]

\[ \text{true,} \]

\[ \text{false,} \]

\[ \text{when} \]

\[ \text{what could} \]

\[ \text{suppose we had} \]

\[ \text{take} \]

\[ \text{use transport:} \]

\[ \text{where} \]

\[ \text{how do you define} \]

\(P\)?
NATREC \((T; \ldots, \cdot, \cdot, 1)\) (a)

\[ a = \text{zero} \quad \text{gives} \quad T \]
\[ a = \text{true} \quad \text{gives} \quad 1. \]

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 \(\text{Id}_{\text{prop}}(\text{zero}, \text{one})\), i.e., \(\text{Id}_{\text{prop}}(\text{true}, \text{one})\) is consistent in ITT.

Two moves:

1. Add an allow "large elims".
2. use natrecc. But what is the type of this particular \(\text{one}\)? Add a type of types, i.e., a universe \(U\).

\[
\begin{align*}
\text{M} & : U \\
\text{NATREC} \quad \text{M} = M' : U \\
\hline
\text{M} & : \text{type} \\
\text{M} = M' : U
\end{align*}
\]

But this causes more problems.

bf. Russell's paradox.

\(\text{M} = \text{U}'\)'s first type theory was inconsistent by \(\text{bf. We had } \text{M} = \text{U}'\).

2018-03-22

Last time mass, we wanted to show that

\[ \text{Id}_{\text{nat}}(\text{zero}, \text{one}) \]

The crucial idea was that we had to define a type by induction on natural numbers:

\[
\begin{align*}
T(a, \text{nat}) & = \text{NATREC}(T; \ldots, \cdot, \cdot, 1) \\
T(\text{zero}) & = T \\
T(\text{one}) & = 1
\end{align*}
\]

This indicates one thing left out by the formal system (cf. Gödel).
But analogously, set theory provides no site for the set of reals. The difference is, set theory has no notion of truth. A formalism will always be incomplete, but at least if 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 a below should be a type is very dangerous:

\[ \Gamma \vdash A_\alpha \text{ type} \quad \Gamma, a : A \vdash b : ? \quad \Gamma \vdash A \text{ type} \]
\[ \Gamma \vdash \text{NA} (A_\alpha, a, b, A) (V) : \text{type} \]

So instead, we

2. Universes: ("maximal") collections of
type (or if you really want, of
codes for types):

\[ \Gamma \vdash A_\alpha : U \quad \Gamma, a : A \vdash b : U \quad U \vdash A : U \]
\[ \Gamma \vdash \text{witness} (A_\alpha, a, b, A) (V) : U \]

well-defined given that we have \( U \).

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 \( U \) are types
( \( \text{"El" decodes the code } A \text{ into a type} \) : 84)
\[ \Gamma + A : U \quad \frac{\Gamma + A = B : U}{\Gamma + A \text{ type} \quad \text{Elim}} \]

If you do this, you need syntax for the codes for your base types, e.g., \( \text{El(nat)} = \text{Nat} \), \( \text{El(Bool)} = \text{Bool} \), etc.

Continuing w/ codes:

\[ \frac{\Gamma + \text{Nat} : U}{\text{Bool}} \quad \frac{\Gamma + A : U \quad \Gamma + B : U}{\Gamma + A + B : U} \]

\[ \frac{\text{Void} \quad \Gamma + A : U}{\Gamma + A : U} \quad \frac{\text{Unit} \quad \Gamma + A : U}{\Gamma + a : A \cdot B : U} \quad \frac{\text{void} \quad \Gamma + a : A \cdot B : U}{\Sigma a : A \cdot B : U} \]

\[ \frac{\Gamma + A : U \quad \Gamma + M, N : A}{\Gamma + \text{El}A(M, N) : U} \]

With universes, we now have the ability to compute types.

\textbf{Note:} You do not have \( U : U \)!

Instead, idea: use an infinite cumulative hierarchy of universes

\[ U_0 : U_1 : U_2 : \ldots \]

\[ \Gamma + M : U_i \quad \text{cumulativity} \]

\textbf{Big question.}

Before, \( I_d^A \) was defined uniformly in \( A \). This was not very rich, because you don't have
1) \( \text{Id}_I : \forall x \in I : x = x \) \( \forall \alpha : \text{Nat}(fa, ga) \to \alpha \)

2) What is \( \text{Id}_I(A, B) \)? I'm trying to answer this. V. Voevodsky came up with some answers to 1).

(Remark: The entire question is illegitimate: just because you want \( \text{Id}_I \) to be "equality" doesn't make it so.)

One could take \( \text{Id}_I(A, B) \) to be isomorphism:

\[ f : A \to B, g : B \to A, \quad \text{s.t. } g \circ f = \text{id}_A \]

What are these "\( = \)"?

But \( (g \circ f)(a) = a \) is too fine, you can't really get this in general.

\( \text{Id}_A(g(f(a)), a) \) is a bit better.

What V. V. came up with is at a high level, isomorphism up to isomorphism.

Idea: explore the generic structure of identity types.

Previously:

\[ \text{refl}_A(M) \quad \text{Id}_A(M, M) \]

\( \text{sym}_A(P) : \text{Id}_A(M, N) \text{ if } P : \text{Id}_A(N, M) \)

\[ \begin{array}{cc}
M & \xrightarrow{P} & N \\
\downarrow & & \downarrow \\
M & \xleftarrow{P^{-1}} & N
\end{array} \]

\[ \text{sym}_A(P) \equiv \text{rev}_A(P) \equiv P^{-1} \]
trans\(_A\) (Q, R): \text{Id}_A (M, P)

If Q: \text{Id}_A (M, N)
R: \text{Id}_A (N, P)

\[
\begin{array}{c}
M \xrightarrow{Q} N \xrightarrow{R} P \\
\text{concatenate the paths}
\end{array}
\]

cat\(_A\) (Q, R) \iff Q \cdot R

Property: \text{refl}_A (M))^{-1} = \text{refl}_A (M) ("inevitable")

Depending on the code for trans\(_A\) (Q, R), we may have
\text{trans}\(_A\) (Q, \text{refl}_A (N)) \equiv Q

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
\text{trans}(\text{refl}_A (N), R) \equiv R.

If you do double induction on the paths, then you only get
\text{trans}_A (\text{refl}_A (M), \text{refl}_A (M)) \equiv \text{refl}_A (M).

What do you get if you do
\[ P \cdot P' \] or \[ P' \cdot P? \]

\[ \# \text{refl}_A (M) \]

in general? How about
\[ P \cdot (Q \cdot R) \] or \[ (P \cdot Q) \cdot R \]
\[(P^{-1})^{-1} \sim P\]
\[P \cdot \text{refl} \sim P\]
\[\text{refl} \cdot P \sim P\]

These are called the groupoid laws.

The expectation is that these laws should hold for identifications. They do hold!

Suppose \(P : \text{Id}_A(M,N)\) moreover \(P^{-1} : \text{Id}_A(N,M)\).

Claim: \(\exists : \text{Id}_{\text{Id}_A(M,N)}(P \cdot P^{-1}, \text{refl}_{A(N)}(M)).\)

Assertion: \(u = \text{Id}_{\text{Id} - \text{elim}}\) on \(P\).

The critical idea is this iterated identification: you have identification of identifications. (Yes, we know all closed instances of \(\text{Id}\) are \(\text{refl}\).)

Error: \((P \cdot \text{refl}) R\)

\(t = 0\)

\(S \Rightarrow P \Rightarrow Q \Rightarrow R \Rightarrow F\)

\(t = 1 \Rightarrow P \cdot (Q \cdot R)\)

The two sides are the top and bottom paths the same? They certainly aren't 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 principal for this. The whole point of univalence was to add enough such deformation.
\[ \mathcal{I} : \text{Id}_{\mathcal{A}(S, F)} \cdot (P \circ q) \cdot R, \text{Id}_{\mathcal{A}(Q, R)} \rightarrow \]

Do it using path induction! \[ J. \]

In short, the groupoid laws hold "up to higher identification". These iterated identifications bring in dimensionality.

- Paths between pt 1
- Paths between 2
- Paths between 3

The structure you get is that of an \( \infty \)-groupoid.

Next

\[ \rightarrow \text{Develop the equivalence axiom } (W) \]

The key technical idea is inspired by the following idea:
When is \( f : A \to B \) a bijection?

\[ f^{-1}(b) \text{ - preimage: }\mathcal{E} = f^{-1}(b) \]

"Fiber of f over b".

\[ f \text{ is a bijection exactly when each of these fibers consists of a single point, i.e., it is contractible.} \]
Homotopy Type Theory (HoTT)

extension of ITT w/two ideas:

1. equiv: types are identified up to equivalence (isomorphism up-to iso)
2. (higher) inductive types that specify points and also paths/identifications

Notation

IdA(M,N) identity type
M=M_A N  identification type
  "path type"

lein form: "path induction"
induction form: refl_A(M) : M=M_A M

Last time we saw the groupoid laws/structure of identity:
  refl_A(M) : M=M_A M
  if P : M=M_A M', then P⁻¹ : M'=M_A M (defn)
  if P : M=M' and Q : M'=M'', then
  P·Q : M=M''  (defn)

s.t. there are all inhabited:
  * refl_A(M) · Q = M=M_A M
  * P · refl_A(M) = M=M_A M
  * refl_A(M⁻¹) = refl_A(M)
  * P⁻¹ · P = refl_A(M')
  * P · P⁻¹ = refl_A(M)
  * (P⁻¹)⁻¹ = P
  * P · (Q·R) = (P·Q)·R
Remarks:

If \( f : A \to B \) and \( P : M =_{A} M' \), then there is an operation

\[ \text{ap} \, (P) : [M] =_{B} [M'] \]  

(which we will later write as \( P(P') \)) such that

\[ \text{ap} \, (\text{refl}_A(M)) = \text{refl}_B(f(M)). \]

Pf: By path induction on \( P \):

\[ J[a, b, \ldots, f(a) = f(b)] \, (a \cdot \text{refl}_B([a \cdot f(a)])(P)). \]

: \( f(M) =_{B} f(M') \)

Though HoTT is not computational, it has a notion of computation via proof reductio à la Gentzen.

Recall

\[ u : \text{A + B type} \quad P : M =_{A} M' \]

\[ \text{tr} \, [a \cdot B](P) : [M/a] B \to [M'/a] B \]

\[ \text{refl}_A(M)(a) = \text{id} \, [M/a] B \]

If \( f : \text{Ta} : B(a) \) and \( P : M =_{A} M' \),
then you would like something like

\[ f(M) =_{B} f(M') \]

\[ [M/a] B \to [M'/a] B \]

The two sides live in different types...

Use "heterogeneous paths", say \( P \)

induces some path \( \text{ap} \, (P) : [M] =_{B} [M'] \) vertical

\[ \text{refl}(P) : [M/a] B \]

\[ [M'/a] B \]

\[ M \quad P \quad M' \]
\[ \text{apd}_f (P) \triangleq \]
\[ \text{J}[a, b, p \cdot \text{tr} [a, B](p)(f(a)) = f(b)](a \cdot \text{refl}_B (f(a)))(p) \]
\[ \text{tr} [a, B](p)(f(\eta)) = [\text{refl}_{\eta B}] \]
\[ f(M') \]
(Also in Hae-H book)

"apd\_f (P) : f(M) = a :: B :: f(M')"

is the notation we use in the dp 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)

\[ \Sigma c : A . \Pi a : A . a =_A c \]

"centre"

identifiers

\[ f : A \to B \text{ is an equivalence iff } "f \text{ is a bijection}, \]
\[ "f^{-1}(b) \text{ is a singleton for all } b : B" \]

\[ \Sigma a : A . f(a) =_B b . \]

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 type"?

Idem "pre-equivalence":

\[ A =_B B \text{ iff } A \preceq B \]

\[ \Sigma f : A \to B . \text{isEquiv}(f) \]

\[ \Pi \text{b : B . is Centro}(f^{-1}(\text{b})) \]
Univalence gives you more than this.

\[ A = \mu B \]

A type \( \mu \) elements:

\[ \text{paths identifying} \]

\[ \text{A type of equivalences} \]

\[ \text{between A \& B} \]

Want to establish some

strong equivalence btw these 2 types.

But we can say even more!

Define idtoequiv \(_{A,B} : A = \mu B \rightarrow A \cong B \)

\[ \text{Id}_{A,B} \quad \text{Eq} \]

using path induction:

\[ \text{Univalence Axiom:} \]

\[ \text{"idtoequiv is an equivalence"} \]

\[ \text{ua}(A, B, E) : \text{Id}_{\text{Eq} (A, B)} \]

\[ E : A \cong B \]

\[ \text{depends on the axiom and is irreducible} \]

and cannot be reduced to refl.

The surprise from a mechanization pov

is that you identify types up to equivalence.

This is the start of a theory of symmetric coercion!

**Problem**

\[ \text{Tranb, p. C} (u, A, \varphi) (\text{ua}(A, B, E)) \equiv ? \]

How do we "speak"/"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 the internalize it.

Idea: \( \text{tr}_{[\mathbb{K}, \mathbb{K}]} \text{tr}(E)) (M; A) \equiv "{E(M)} \)

should come out to be true. And indeed, in the model of the case 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. We "Martin-Lof's" marvels onto moves where he hardly moves and everybody is on the floor.

(Higher) Inductive Definition.

Idea: specify the free type on some generators. In group theory, you have free groups on generators satisfying some relations, but in \( \text{tr}_{[\mathbb{K}, \mathbb{K}]} \text{tr}(E)) \), you don't have just relations.

**Example:** Interval type \( I \)

\[
\begin{align*}
0 & : I & \text{"points"} \\
1 & : I & \\
\text{seg} & : 0 = 1 & \text{"line"}
\end{align*}
\]

So to speak, there's additional stuff brought in by the fact that you're generating the free \( \text{co-groupoid} \) 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 \( \text{tr}_{[\mathbb{K}, \mathbb{K}]} \text{tr}(E)) \), but also all of the other stuff.

Next time will derive the eliminator. ersum. \( \frac{94}{94} \)
Univalence

Implies that equal types in a universe are exactly equivalent types

\[ \text{IdtoEquiv: } A =_{B} B \Rightarrow A \equiv B \]

\[ \Rightarrow \text{ definable by path induction} \]
\[ \left[ \text{Axiom: a way to build} \right. \]
\[ \text{paths in } U \text{ out of equivalence} \]
\[ \text{Eqv}(A, B, E) \text{ is a "real" identification} \]

Claim (V.V.): univalence suffices for function extensionality.

Def (total path space)

\[ \text{Paths}(A) = \sum_{a \in A} \left( \sum_{b \in A} \left( a \equiv_{A} b \right) \right) \]

\[ \begin{array}{c}
\text{based path space at } a \\
\text{star at } a \\
\text{equivalence class at } a
\end{array} \]

Remark: that path spaces are contractible singletons.

Fact: \[ A \equiv \text{Paths}(A) \]

Indeed, every ele of \( A \) has a path space and every path space contracts to an equivalence ele of \( A \).

And so by univalence, \[ A \equiv \text{Paths}(A) \]

Remark: \( J \Rightarrow \text{transport + contra bend} \)

But also \( J \equiv \text{path space} \)

See the note by C. Anicic for details.
**Def:** \[ \text{Htpy}(A, B) \triangleq \sum_{f, g : A \rightarrow B. f \sim g} \]

where \[ f \sim g \triangleq \Pi a : A. f(a) =_B g(a). \]

**Fact:** \[ (A \rightarrow \text{Paths}(B)) \simeq \text{Htpy}(A, B) \]

**Proof:** \[ (\Rightarrow) \quad \forall F. \quad \lambda A. F(a) \cdot 1, \lambda a : A. F(a) \cdot 2, \lambda a : A. F(a) \cdot 3 \]

\[ \quad \left( \iff \right) \quad \lambda H. \quad \lambda a : A. \langle (H \cdot 1)(a), (H \cdot 2)(a), (H \cdot 3)(a) \rangle. \]

You can show these are mutually inverse.

**Remark**

\[ \text{Paths}(A \rightarrow B) \simeq \text{Htpy}(A, B) \]

expresses function extensionality.

\[ \text{Htpy}(A, B) \rightarrow \text{Paths}(A \rightarrow B) \]

is easily definable.

\[ \left( \iff \right) \begin{align*}
\text{Paths}(A \rightarrow B) &= \{ a : A \rightarrow B \} \\
\text{Htpy}(A, B) &= \{ \text{htpy}(a) \rightarrow B \}
\end{align*} \]

These are equalities.

**Cor:** Univalence entails function extensionality.

**Inductive Types**

Key idea: inductive types are characterized by a "merging out" property—initial object.

**Eg:** natural numbers

\[ \Gamma \vdash \text{add} : \forall n, m : \mathbb{N}. n + m \]

\[ \Gamma \vdash \text{zero} : \mathbb{N} \]

\[ \Gamma \vdash \text{success} : \mathbb{N} \] 

You say this is the least claim by saying if you want to merge out, it's suff to consider only 0 and \( s(0) \).
\[ \Gamma \vdash M : \text{Nat} \quad \Gamma \vdash A \text{- type} \\
\Gamma \vdash M_0 : A \\
\Gamma, a : A \vdash M_{\text{rec}} : A \\
\Gamma \vdash \text{natrec}(M_0, a ; M_{\text{rec}})(M) : A \]

Dependently if you want:

\[ \Gamma \vdash M : \text{Nat} \quad \Gamma, a : \text{Nat} \vdash A \text{- type} \\
\Gamma \vdash M_0 : A(0) \\
\Gamma, a : A(n) \vdash M_{\text{rec}} : A(n) \\
\Gamma \vdash \text{natrec}(M_0, a ; M_{\text{rec}})(M) : A(M) \]

This second elim is definable from the 
simply-typed one + \( \Sigma \)-types.

\[
\begin{align*}
\text{natrec}(M_0, a ; M_{\text{rec}})(0) & = M_0 : A(0) \\
\text{natrec}(M_0, a ; M_{\text{rec}})(s(m)) & = \\
\text{let } M, \text{natrec}(\text{rec}(M), n, a ; M_{\text{rec}}) : A(n) & = \end{align*}
\]

Claim: probate uniqueness up to unifiers.

"Higher" Inductive Types

Idea: \( A \) is somehow coupled to \( \text{Id}_A(-,-) \)

\[
\begin{align*}
\text{Id}_A(-,-) & \rightarrow \\
\text{Id}_{\text{Id}_A(-,-)}(-,-) & \\
\end{align*}
\]

Therefore we can consider a certain form of definition that populates not only the type \( A \) but also its companion \( \text{Id}_A \) types.

"The beauty of formal type theory is that you can write down whatever."

Example: "The interval" \( I \), which is inductively defined as follows:

\[
\begin{cases}
\Gamma \vdash 0 : I \\
\Gamma \vdash 1 : I \\
\Gamma \vdash \text{seg} : \text{Id}_I(0,1)
\end{cases}
\]
Fact: \( I \) is contractible.

Consider some \( h: I \to A \)

The eliminator is:

\[
\frac{\Gamma \vdash M : I \quad \Gamma, i : I \vdash A \text{ type}}{
\Gamma \vdash M_0 : A(0) \quad \Gamma \vdash M_1 : A(1)}
\]

\[
\frac{\Gamma \vdash M : I \quad \Gamma \vdash A \text{ type}}{
\Gamma \vdash M_0 : A \quad \Gamma \vdash M_1 : A}
\]

\[
\Gamma \vdash \text{Id}_A(M_0, M_1)
\]

\[
\Gamma \vdash \text{Inec}(M_0, M_1, M_{seg})(M) : A
\]

\[
\text{Inec}(-)(0) = M_0, \\
\text{Inec}(-)(1) = M_1,
\]

\[
\text{op}(\text{Inec}(-)(\text{seg})) = (M_0 =_{A,M_1}) M_{seg}
\]

\[
\text{Computation here}
\]

\[
\frac{\Gamma \vdash M : I \quad \Gamma, i : I \vdash A \text{ type}}{
\Gamma \vdash M_0 : A(0) \quad \Gamma \vdash M_1 : A(1)}
\]

\[
\Gamma \vdash M_{seg} : M_0 =_{\text{seg}} M_1,
\]

\[
\Gamma \vdash \text{Inec}(M_0, M_1, M_{seg})(M) : A(M)
\]

Where \( M_0 =_{\text{seg}} M_1 \) means (by def.)

\[
\frac{A(0) \xrightarrow{M_{seg}} M_1}{\xrightarrow{\text{Id}_A(M_0)} A(1)}
\]

\[
0 \to 1
\]
and 
\[ \text{I-rec}(-)(0) \equiv M_0 \]
\[ \text{I-rec}(-)(1) \equiv M \]

apply \( \text{I-rec}(-)(1) \) to \( \text{seg} \)
\[ \text{M}_{\text{seg}} \to M \]

heterogenous partial space.

Jack

Paths \( (A) = \mu \text{I} \to A \)
\[ \exists a, a', a \leftarrow a \]

If
\[ \forall a, a', p. \exists h. \text{I-rec[A]}(a, a', p)(i) \]
\[ \forall h: I \to A. \langle h(0), h(1), \text{ap}_h(\text{seg}) \rangle \]

+ check there are mutually inverse. \( \square \)

Paths \( (A \to B) = \mu \text{I} \to (A \to B) \)

\[ = \mu (I \times A) \to B \]

\[ = \mu (A \times I) \to B \]

\[ = \mu A \to (1 \to B) \]

\[ = \mu A \to \text{Paths}(B) \]

\[ = \mu \text{H}_{\text{Py}}(A, B) \]

So the internal also gives your function extensionality.

Rmk: This is the standard textbook version of homotopy.
"Higher inductive define idea is to simultaneously define (by specifying generators) types $A$, $I$."

Motivation: $A$ is "coupled" with its path spaces.

1) We introduced the abstract interval $I$. $\emptyset : I \xrightarrow{\text{seg}} I_0 \xrightarrow{\text{lin path}} I_1$.

$I$ draws a line in $A$.

$I^n \rightarrow A$: n-cube in $A$ for $n \geq 0$.

Eliminations for $I$.

\[
\begin{align*}
\Gamma &+ M : I & \Gamma, i : I + C & \text{type} \\
\Gamma &+ \text{Mo} : \text{Co}/i/C & \Gamma, h : I +/i/C & (\text{seg}, \text{lin}/C(i)(\text{seg})) \text{Mo} \\
\Gamma &+ \text{H} : \text{I}/i/C & \Gamma, h : \text{I}/i/C & (\text{seg}, \text{lin}/C(i)(\text{seg})) \text{H} \\
\Gamma &+ \text{M} : \text{Mo} = i.e. M, & \Gamma, i : \text{I}/i/C & \\
\Gamma &+ \text{I-ind}[i.e.] (\text{Mo}, M, \text{H}) & (H) : \text{[H}/i/C}$
\end{align*}
\]

1) $\text{I-ind}[i.e.] (\text{Mo}, M, \text{H}) (0) = \text{Mo} : \text{Co}/i/C$

\[
1 = M, : \text{I}/i/C.
\]

2) $\text{apd} (\text{I-ind}[i.e.] (\text{Mo}, M, \text{H})(-)(\text{seg})) = M$

5) "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."

\[100\]
Circle $C$ aka $S^1$ (the soft loop)

Base: $C$
Loop: $I \cup \text{Close, base}$.

\[
\begin{align*}
\Gamma : & \quad C + C \quad \text{type} \\
\Gamma + M : & \quad M_{\text{base}} : C \cap \text{base} \\
\Gamma + M_{\text{loop}} : & \quad M_{\text{base}} = \text{loop} \quad M_{\text{base}} \\
\Gamma + C_{-\text{ind}}[C(M_{\text{base}}; M_{\text{loop}})(M)] : & \quad [H \in \{C\} \\
\text{base} \in C_{-\text{ind}}[C(M_{\text{base}}; M_{\text{loop}})(M)] : & \quad \text{base} \in C_{-\text{ind}}[C(M_{\text{base}}; M_{\text{loop}})(M)] \\
\text{base} \in C_{-\text{ind}}[C(M_{\text{base}}; M_{\text{loop}})(M)] : & \quad \text{base} \in C_{-\text{ind}}[C(M_{\text{base}}; M_{\text{loop}})(M)] \\
\end{align*}
\]

You could have written $\Gamma + M_{\text{loop}} : M_{\text{base}} = \text{base} \cap C_{-\text{ind}}[C(M_{\text{base}}; M_{\text{loop}})(M)] \text{ at } \circlearrowleft$. 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"

$\Omega(A; a_0) \cong \frac{a_0 = a \circ a_0}{I a(a_0, a_0)}$

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-truncated" $\Omega(A; a_0)_{[0]}$, which squashes down all of the higher-dimensions.

The truncated loop space of $C = (I, +, 0)$ 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:
   - programmable type systems (coercion, aka homotopy)

2. Martin-Löf's papers on judgments (epistemically 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:

- Hilbert systems which have only implication
- τ vs. combinators

The entire idea is $A \text{ true} \rightarrow B \text{ true}$ (entailment)

This is a "ninja move."

The generalization of entailment is something similar for universal quantification:

$$\forall x \ (A \text{ true}) \quad \text{(generality)}$$

$$\forall x A \text{ true} \quad \text{(internalization)}$$

Constructivism (the idea that the evidence for the truth of a prop is a thing)

Suggests consolidation:

"hypothetical-general judgments"

$$\exists x : A, \quad \forall x : A \rightarrow P : A$$

variables
This emphasizes the centrality of variables and what they mean in type theory:

1) Formal:
   \[
   \left\{ \begin{array}{c}
   \Gamma \vdash M : A \\
   \Gamma \vdash M = N : A
   \end{array} \right\} \Rightarrow \Gamma \Rightarrow M = N : A
   \]

   Semantic placeholders

   "admissibility"

They both share the structural properties and present a consequence rule:

(compositor) Substitution/transitivity

\[
\Gamma, x : A \vdash N : B \\
\Gamma \not\vdash M : A \\
\hline
\Gamma \not\vdash [M/x]N : B
\]

(reflexivity) variable

Weakening/"extra variables"

Contraction/"duplicate variables"

Aside: The first version of Church's \( \lambda \)-calc required you to use all vars, so you couldn't write \( \lambda x. y. x \). This was the \( \lambda 1 \) and \( \lambda 1 2 \) case.

He then changed his mind and then you get the \( \lambda 2 \) calculus (aka: \( K = \lambda 2, \lambda y. x \)).

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.

If you can't just check in copro to \( \bot \), because it destroys
the computational contents 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 dimension. The dimension in HoTT corresponds to the number of iterations of \( \text{Id} \).

\[ \begin{array}{c}
\text{A} \\
\text{A} \\
\text{A} \\
\text{A} \\
\end{array} \]

We want to give these names:
- points
- lines
- squares
- cubes

\[ \ldots \]

We want judgments that capture these.

Quine: "No entity without identity"

\[ \begin{align*}
M & = M' \in A & \text{type } & \text{points} \\
M & = M' \in A & \text{type } & \text{lines} \\
\end{align*} \]

We want to be able to answer this, we must first answer who or of what.

\[ \begin{align*}
A & = A' \text{ type } & \text{of points} \\
A & = A' \text{ type } & \text{of lines} \\
\end{align*} \]

The thing to recall is that families of types \( F[A] \) are central to \( \text{DTT} \).

\[ F[M/a] \text{ instance} \]

The point is that lines in \( A \) enduce lines between instances.

\[ \begin{array}{c}
M \\
M' \\
\end{array} \rightarrow F[M] \text{ coercion} F[M'] \]

\[ \begin{array}{c}
\text{abstract} \\
\text{identification} \\
\text{transport} \\
\end{array} \]

The thing you care about at the end of the day is how the classifier 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.

\[ M \cong M' \& A @ 1 \] is a line in lines, a heterogeneous path.

Back to the centrality of vars.

Idea: Use Cartesian coordinates.

Specify the dimensionality by contexts of variables that "range over" an interval.

\[ A' = A_x \text{ type } [x_1, \ldots, x_n] \quad (n > 0) \]

\[ M' = M_x \text{ type } [x_1, \ldots, x_n] \]

\[ 0 \leftrightarrow 1 \]

\[ \text{types} \]

\[ A_0 \xrightarrow{A_x} A_1 \]

"\( x_1, \ldots, x_n \in A\)" negatively means "\( x_1 : I \ldots x_n : I + A\)" c.e., "\( x : I^n \& A\) type".

So you get \( n \)-cubes.

There's no notion of distance: you're either one of the endpoints, or never "half-way" along the interval. Oddly enough, just like plugging in a real point causes the Cost to zero, so can do "plugging in a variable \( y/x \)."

The lines in our setup will tell us how to coerce.
Dimensionality and dimensionless

\[ T = x_1, \ldots, x_n \ (n \geq 0) \]

de "Cartesian cords in n-space"

Let \( \Psi : T' \to T \) be a subset for dimensionless. If \( \Psi, T = x_1, \ldots, x_n \), then \( T', \Psi(x_1) \)dim

\[ T' = O(1) \mid x \]

Dimension contexts are structural

\[ \begin{align*}
\overset{\text{f, x}}{\Psi} & \overset{\text{f}}{\Psi} \\
\overset{\Psi, x, y}{{\Sigma}_x} & \overset{\Psi, x, y}{{\Sigma}_y} \\
\overset{\Psi, x, y}{{\Sigma}_y} & \overset{\Psi, x, y}{{\Sigma}_x}
\end{align*} \]

If \( M \text{ tm } [\Psi] \) and \( \Psi: T' \to T \text{ th, } M < \Psi > \text{ tm } [\Psi] \).

Variance functional action of \( \Psi \) on terms "pullback"

This preserves identity and composition "on the nose"

\[ M < \text{id} > = M \]
\[ M < \Psi_1 \cdot \Psi_2 > = (M \cdot \Psi_1) \cdot \Psi_2 \]

Can give a presheaf semantics

- Kripke semantics where worlds = dimensions
Substitution of 0, 1 picks out "left"
\[ M(0/x) \rightarrow Mx \rightarrow M(1/x) \]
"left" \hspace{1cm} "right" \\
\[ x: 0 \rightarrow 1 \]

We have a cubical programming language
\[ M \text{ Val}_{!} \rightarrow M \]
\[ M \rightarrow M' \rightarrow M'' \]
\[ \Pi_{\equiv} \in \text{Tm}(\Pi) \times \text{Tm}(\Pi) \]
Could omit the dimension etc if wanted.
Note these rules do not commute with sub list in general!
\[ M \text{ Val}_{!} \text{ but not } M(0/x) \text{ Val}_{!} \]
\[ M \rightarrow M' \text{ but not } M(0/x) \rightarrow M'(0/x) \text{.} \]

eg) circle C in computation HT

base val
loop \rightarrow loop \rightarrow base
loop \rightarrow loop \rightarrow base

In some Fctx (Π):
\[ C \text{- elin}(C. \text{C})(M \text{ Val}_{!} \cdot x.M \text{ Val}_{!})(M) \]
\[ \rightarrow C \text{- elin} \rightarrow (M') \text{ if } M \]
\[ \mu \text{ be careful}\]
\[ \Pi_{\equiv} \]
\[ x: F \rightarrow \alpha \]
\[ \rightarrow \alpha \rightarrow (\text{loop}) \rightarrow M \text{ Val}_{!} \]
\[ \rightarrow \alpha \rightarrow (\text{loop}) \rightarrow H \text{ Val}_{!} \]
Remark that these reductions work at all dimensions.

Note: Substituting into a loop changes a value into a non-value.

\[ \text{loop}_x \text{ val} \rightarrow \text{base} \]

Also note that reduction doesn't commute in subshift!

\[
\begin{align*}
\text{c-elim} (\text{loop}_x) & \rightarrow \text{val} \rightarrow \text{base} \rightarrow \text{base} \\
& \rightarrow \text{base} \\
& \rightarrow \text{base} \\
& \rightarrow \text{base}
\end{align*}
\]

and if you then do the sub-shift \( \llbracket \text{M} \text{loop}_x \rrbracket \),
you have a semantic confluence, but not a syntactic confluence, and
you cannot get a coherence via the \( \triangleright \) sem. The equality in a semantic notion via the type...

What does the judgment \( \text{M} \text{A} \llbracket I \rrbracket \) mean?

(Uninformal account first)

What \( \llbracket I \rrbracket \) is means \( \text{M} \llbracket A \rrbracket \) and \( \text{M} \llbracket \text{loop}_x \rrbracket \) so that

and \( \llbracket \text{loop}_x \rrbracket \) is a value of type \( \text{A} \) but \( \llbracket I \rrbracket \) equal?

(it's all about process running and specifying their value)

A. First cut in \( \llbracket I \rrbracket \):

**Case:**

\[
\begin{align*}
\text{A} & \llbracket \text{A}_0 \rrbracket, \quad \text{M} \llbracket \text{Mo} \rrbracket, \quad \text{Mo is a} \text{-value of type} \text{A}_0 \\
\Rightarrow & \text{Need to explain where the} \text{-values are.}
\end{align*}
\]

But consider:

if \( \text{loop} \in \text{C}[x] \) then \( \text{loop} \in \text{C}[*] \)?

You want this to be true:

\[
\begin{align*}
\text{base} \quad \text{loop} \quad \text{base} \quad \text{loop} \\
& \text{loop} \quad \text{base} \quad \text{loop} \\
\end{align*}
\]

But you need to make it
The meaning of must account for face maps ( intim to get "side"

Desideratum #2! (generalize # 1)

If \( \forall A_0, \text{for all } f : T' \rightarrow T \) ("T' aspects")

\[ M \in A_0 \{ T' \} \]

i.e., \( M \in A_0 \{ T \} \) and \( M_0 \) is a \( T' \)-value of \( A_0 \).

e.g. \( \text{loop}_x \in C [X] \) implies \( \text{loop}_0 \in C [\cdot ] \)

\( \text{loop}_1 \in C [\cdot ] \)

Closure: Suppose you have a cube

\[ M_2 \]

\[ \text{versus } M (1,0/2,4) \in M_3 \]

\[ M_1 (0/4) \in M_2 \]

Is \( M_2 = M_3 \)?

Want to demand that they be exactly equal in whatever type they're i.e., \( M_2 \equiv M_3 \in A_0 \{ x \} \)

Nothing forces this could write down bad programs. But lots of "bad" programs exist and you just decide not to give them a type. Your types are supposed to the specifications of "good" programs, do you just design your typed system to ensure \( \Box \) holds.
Desideratum #3: COHERENT ASPECTS

Let \( M \in \mathcal{A}[\mathcal{I}] \) be

\[
\begin{align*}
\& \quad \Lambda \subseteq A \subseteq M \cup M_0 \\
\& \quad \forall \psi : \mathcal{I} \rightarrow \mathcal{I}, \quad \forall \psi : \mathcal{I} \rightarrow \mathcal{I} \quad \text{then} \\
\& \quad M \psi \subseteq M_0, \quad M, \psi \subseteq M, \quad M(\psi, \psi) \subseteq M
\end{align*}
\]

Then \( \exists \mathcal{A}_{M_2} : M_2 \in \mathcal{A}_0[\mathcal{I}_2] \).

This is still not enough! What about DEPENDENCY?

- Types can vary in a dimension!

\[
F[M] : A \text{- indexed family of types} \quad F[M], \quad \text{where } M \in \mathcal{A}[\mathcal{I}] \text{ is a type}
\]

varies in the dimension!

\[
F[M] \langle x/x \rangle = F[M \langle x/x \rangle]
\]

- Dep Des #1:

\[
\begin{align*}
\& \quad A \psi : \mathcal{I} \rightarrow \mathcal{I} \\
\& \quad M \psi \subseteq A \psi [\mathcal{I}]
\end{align*}
\]

i.e., \( M \psi \subseteq M_0 \) and \( M_0 \) is a \( \mathcal{I} \) value of \( A \).

The picture for coherent aspects becomes

\[
M \subseteq \mathcal{A}[\mathcal{I}_2, \mathcal{I}_2]
\]

The front face of \( M \) should inhabit the top right edge of \( A \), etc.
It all has to cohere.

Day: Dec #2: Coherent aspects in higher dimension

If $M \in A(\mathcal{E})$ then

$A \cdot A, M \cdot M, o$

If $\psi : \mathcal{E} \to \mathcal{E}, \psi_2 : \mathcal{E} \to \mathcal{E}$

$M(\psi, M), M(\psi_2, M), M(\psi, A), A(\psi, A)$

Then

$M_{12} = M_{2 \cdot 6} A_2 (\mathcal{E})$

and

$A_{12} = A_2$ type.

Aside. Equality is prior to path. Because you can "plug" the paths "fit together" to form another squares if you have a factor of equality of judgments.

Hypothetical general judgments

What's the meaning of

$a : A \Rightarrow \text{NEAR} \neq B [\mathcal{E}]$?

Want this to be a mapping, so you could say:

If $M = M' \in A(\mathcal{E})$ then $N[M]a = N[M'/a]$.

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 works for just the current dimension, but at all accessible dimensions so that you respect paths. So you actually want:

\[ F : \mathcal{F} \to \mathcal{F} \]

\[ \psi \models M \iff M' \in \mathcal{A} \text{ then } \]

\[ \psi \models [M/a] = [\psi \models [M'/a]] \vee [B \models [M/a] = [B \models [M'/a]]] \]

So the "and" behaviour from NotT is built-in. \[ F \] could be \[ F \times \]

"So if you have a line between \[ F \]-cubes of type \( A \), you get one between \[ F \]-cubes of type \( B \)."

In summary: make everything behave properly as presheaves.
Hypothetical Judgments

1) \( (a : A \Rightarrow B) [F] \) (families)
2) \( (a : A \Rightarrow N \in B)[F] \) (maps)

Namely, (incorrectly), these are "separate facts" at each \( F \).

1) \( M = M' \in A[F] \) then \( B[M/A] = B[M'/A][F] \)
2) \( N[M/A] = N[M'/A] \subseteq B[M/A][F] \)
\( (= B[M'/A]) \)

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 a Kripke semantics point of view, you observe that the above is insufficient to maintain truth at the relevant future worlds.

More precisely:

\[ \forall \psi : F' \rightarrow F \text{ "aspects" (false, diagnose, degenerate, ...) } \]

if \( M = M' \in A[F'] \), then
\[ N[M/A] = N[M'/A] \subseteq B[M/A][F'] \]
\( (= M[A]) \)

Our possible worlds are the \( F \) and the reachability relation is given by the aspects.

In particular, consider what happens when \( \psi \) is a subset
weakening \( \psi : F, x \rightarrow F \).
\[ \{ \text{ if } M = M' \in A[F, x] \} \]
\[ \{ \text{ then } N[M/A] = N[M'/A] \subseteq B[M/A][F, x] \}

i.e., \( N \) must take lines to lines and respect paths. \( A, N, \) and \( B \) do

\( \psi \) might not involve \( x \), but \( M \)
and \( M' \) might (and so be lines), so \( N \) takes times 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 indeterminates, replaced by dimensions in vars. They do not range over closed terms. This is similar 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.

\[ \text{Note: } M \in A[F, x] \quad \Rightarrow \quad M \langle 0/x \rangle \in A \langle 0/x \rangle \]

There is more to a line than just its endpoints.

Let's define some types

(Warning: This will be both boring and wrong.)

\[ \text{eg } \text{Bool} = \text{Bool}[F] \quad (\text{at any dimension } F) \]

\( \text{Bool} \) is a degenerate cube on \( F \) (doesn't depend on \( F \)).
true = true ∈ Bool [T]
false = false ∈ Bool [F]

\[\text{trueVal} \quad \text{falseVal} \quad \text{BoolVal}\]

\[M_0 \rightarrow M'\]

if \([a, C])(M; M_0; M) \rightarrow if \([a, C])(M; M_0; M)\]

Assuming these transitions are stable under time, this doesn't always hold.

**Fact:** If \(a: \text{Bool} \rightarrow \text{CType} [T]\)
and \(M \in \text{CType} [T]\)
and \(M_0 \in \text{CType} [T]\)
then \(\forall \alpha. \alpha \in \text{CType} [T]\)

Proof is similar as before.

\(\forall \alpha. \alpha \in \text{Bool} \rightarrow \text{CType} [T] \text{ iff} \left(\forall \alpha. \alpha \in M \rightarrow \text{CType} [T]\right)\)

Note the apparent circularity. This is not a problem because of the constructor used on the operator and we're working in its full form.

\(a: A \rightarrow B \text{ type } [T] \text{ iff} \left(\forall \alpha. \alpha \in a: A \rightarrow B \text{ type } [T]\right)\)

We're working in the fixed point that defines the types, in a constructor that generalizes the one we saw earlier.
Inductively defining some formal object.

Remark. Because functions internalize mappings (see \( \Theta \)), it is sufficient to get mappings right to get functions right.

In HoTT, we needed \( \Pi \) and \( \Sigma \). Here, we don't, because there's an intrinsic notion of action on \( \pi \) that built in to the system.

\[ \text{Eq} \ (M, N) \quad \text{type} \ [F] \quad \text{(intuition: internalize equality)} \]

\[ \text{iff} \quad A \quad \text{type} \ [F] \]
\[ M \in A \ [F] \]
\[ N \in A \ [F] \]
\[ P \in \text{Eq} \ (M, N) \ [F] \]

\[ \text{iff} \quad P \text{ refl}(R) \quad \} \quad \text{inductive def of} \]
\[ M = R = N \in A \ [F] \]

Notice:

"equality reflection": if \( P \in \text{Eq} \ (M, N) \ [F] \)
then \( M = N \in A \ [F] \).

It internalizes exact equality.

Fact. \( \text{Eq} \ (-,-) \) is the least reflexive rel in

This can be internalized as an elim form, giving you the \( F \) you
Define

\[
\{ \text{for } \alpha : A, \beta : A, \gamma : C, \delta : (A \times B) 
\}
\]

\[
\text{iff } \alpha = \beta \Rightarrow \gamma \in C[a, \alpha, \text{refl}(\alpha)] \quad [\text{E}]
\]

\[\text{Pe Eq}_A (M, N) \quad [\text{E}]
\]

Then \( \text{J}[a, b, c, C](a, Q)(P) \in C[M, N, P] \).

This says \( \text{Eq}_A \) is the least refl rel in:

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 \( \mathfrak{M} \):

\[\mathfrak{M} \in C[M, (N, P)] \quad [\text{E}]
\]

\[Q[M, \alpha] \in C[M, (M, \text{refl}(M))] \quad [\text{E}]
\]

Want to see the underlined \( \mathcal{I} \) as equal members of

\[a : A \times \text{Eq}_A (M, \alpha)
\]

\[(M, \text{refl}(M)) \uparrow \quad [\text{E}]
\]

\[(N, P) \downarrow \quad [\text{E}]
\]

Because of equality refl: \( M = N \) by \( P \).

So \( \text{refl}(M) = P \) by \( (+) \), so the pairs are equal. So \( \mathfrak{M} \) is \( Q[M, \alpha] \).

One can make the whole same arg in part.

Critical: \( \text{Pe Eq}_A (M, N) \) iff \( P \subseteq \text{refl}_A (R) \) in

(Originally, \( \text{Eq} \) & \( \text{I} \) were confused because...
"Path Type"


Code: internalize the path structure, and go back & forth between paths and points, similar to how we internalize the mapping structure as functors.

Path \((M, N) \text{ type } [F]\)

```
A type \([F, x]\)
```

\(\text{def} M \in A \langle 0/x \rangle\)
\(N \in A \langle 1/x \rangle\).

P : Path \((M, N) \in [F] \text{ iff (def)}\)

\(P \triangleright <x, Q> \text{ ("quoting" a line)}\)
\(Q \in A \langle F, x \rangle\)
\(Q \langle 0/x \rangle = M \in A \langle 0/x \rangle \text{[F]}\)
\(Q \langle 1/x \rangle = N \in A \langle 1/x \rangle \text{[F]}\)

Here, paths end at their endpoint on their slice. You could alternatively have a "Path Type" in PathFun that doesn't mention them.

Define \(M @ x \rightarrow M' @ x\) when \(M \rightarrow M',\)
\(<x, Q>@x \rightarrow Q@x\)

\(\text{Set: } \text{def} P \in \text{Path}_{x,A}(M, N)\) then

\(P@0 = M \in A \langle 0/x \rangle\)
\(P@1 = N \in A \langle 1/x \rangle\)
We saw a cubical semantics for □:
\[ \Gamma' \Rightarrow A = B \ [\square] \]
\[ \Gamma \Rightarrow M = N \triangleleft A \ [\square] \]

satisfying a presheaf condition
if \[ \Gamma[\square] \] and \[ f : \Gamma' \rightarrow \Gamma \], then \[ \Gamma' [\square] \].

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 unique from all accessible worlds.

\[ \text{Eq}_A (M, N) \triangleleft \ [\square] \]
\[ \text{refl}(R) \in \text{Eq}_A (M, N)[\square] \iff M = R = N \triangleleft A[\square]. \]

This ends up (as we'll see) only allowing a "pre-type." No need for an elim form, \[ \forall c \in \text{Eq}_A (M, N)[\square] \iff M = N \triangleleft A[\square]. \]

\[ \text{Path}_{x,A} (M, N) \triangleleft \ [\square] \]
\[ \langle x.P \rangle \in \text{Path}_{x,A} (M, N) \iff \]
\[ P \in \text{A}[\square, x] \]
\[ P < 0/z > = M \in \text{A}<0/z> [\Psi] \]
\[ P < 1/z > = N \in \text{A}<1/z> [\Psi] \]

When \[ x.A \] is degenerate, then this looks a lot like \[ \text{Id}_A (M, N) \] in HoTT
(Homogeneous lines) = \[ \text{Path}_{x,A} (M, N) \]

Define \( M@r \) as follows:

\[ M \mapsto M' \]
\[ M@r \mapsto M'@r \]
\[ \langle x.P \rangle@r \mapsto P(x/r) \]
Fact: If \( \text{Path}_{z,A} (M, N)[x] \) then
\[
\begin{align*}
\text{P} @ z & \in A_z \text{[F,x]} \\
\text{P} @ 0 & \equiv \text{M} @ [x] \\
\text{P} @ 1 & \equiv \text{N} @ [x]
\end{align*}
\]

Though \( Eq_A[M] \) has no elim form, \( J \) is "definable" for \( Eq_A \).

\[
J[\text{a,b,c,C}](\text{a,C})(\text{refl(M)}) \rightarrow Eq_M[a].
\]

Under suitable conditions we've seen before we have
1) \( Q[M,a] \in C[M,M,\text{refl(M)}]a,b,c \)
2) \( b/c \quad P \in Eq_A(M,N)[F] \), we know
\[
\begin{align*}
\text{a) } & P \equiv \text{refl}(M) \in Eq_A(M,N)[F] \\
\text{b) } & M = N \in A[1,F]
\end{align*}
\]
3) We can "transport" \( Q[M,a] \) from \( C[M,M,\text{refl}(M)] \) to \( C[M,N,P] \); by a remark from last time, these two types are equal/identical.

Fact: \( \text{Path}_{A}(-,--) \) 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 \( C[M,M,\text{refl}(M)] \) --- \( C[M,N,P] \).

(Understanding this fact was a
very important development)

Kan conditions arise from the following:

1. Of what use is a path?
   
   the only use of paths within types
is to define paths between types.

2. What paths are there?
   - equivalence
   - paths induced by families
   - closure conditions

We saw the following fake def'n of $C$:

\[
\begin{align*}
C & \text{ type } \mathcal{V} \\
\text{base} : C \mathcal{V} & \\
\text{loop} : C \mathcal{V,2} & \Rightarrow \text{base} \text{ loop base} \\
\text{loop} \Rightarrow \text{base} & \Rightarrow \text{base} \\
C \text{-elim} C \mathcal{V} (C \text{-base} x M \text{-loop}) & \Rightarrow M \text{-base} \text{ (loop x) } \Rightarrow M \text{-loop}
\end{align*}
\]

This runs!

This works and you can prove various things w/ it, but it isn't what you want: there are no elements loop' loop' or loop' capturing going around the loop multiple times or in the opposite direction. You want this to be the first type on these generators, that because this is inductively defined. We'll see what we mean by this later, but these are the closure conditions.

We'll see a nice interplay between the pos & neg types: the negative type will already have all of the closure structure right; pos type will need to be augmented.
Every inductive type (including \( \mathbb{C} \)) is positive.

**Idea**

Lines represent "identifications" of their end points (it is misleading to call them equal). Identifications are witnesses to interchangeability.

\[ M_0 \quad M_x \quad M \]

1) They should be syma terms if they are to be thought of as identifications:

\[ M, \quad M_x \quad M_0 \]

\[ M_0 \quad M_x \quad M_1 = \exists N_0 \quad N_x \quad N \]

adjacent

(For 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.)

\[ M_0 \quad M_x \quad N_x \quad N_1 \]

2) Substitutive - coercion prop.

\[ F[M_0] \quad F[M_x] \quad F[M_1] \]

witness interchangeability.
Question

If \( p \in \text{Path}_A (M,N) \),
\( q \in \text{Path}_A (N,P) \),
then \( p \circ q \in \text{Path}_A (M,P) \)?

Define \( F(-) = \text{Path}_A (M,-) \).

\( \rightarrow \) an \( A \)-indexed family of types.

\( a : A \rightarrow \text{Type} \)

(Notice to use the action of \( F \) on paths)

\[ F[ q \circ x] \text{ type } [x] \]

Looking at the presheaf condition:

\[ F[ q \circ x] \text{ type } [x,x] \] is a line

\[ F[ q \circ x] (0,x) = F[ q \circ 0] = F[N] \]
\[ F[ q \circ x] (1,x) = F[ q \circ 1] = F[P] \]

\[ \Rightarrow \]

\[ F[N] \xrightarrow{q \circ x} F[P] \]

\( \text{Path}_A (M,N) \xrightarrow{q \circ x} \text{Path}_A (M,P) \)

We have a line \( \text{Path}(M,N) \rightarrow \text{Path}(M,P) \)
induced by \( \circ q \) a line \( N \rightarrow P \).

"Ought to be that we can transfer
\( p \in \text{Path}_A (M,N) \) along \( \text{Path}_A (M,q \circ x) \)
along \( \text{Path}_A (M,P) \) to obtain
\( p \circ q \in \text{Path}_A (M,P) \).

Claim: the order for transitivity to hold,
there ought to be correct.
Coerce from 0 to 1.

\[
\text{coerce } x \rightarrow 0 \rightarrow 1 \quad \text{in Path} \rightarrow \text{Path}_A(M,N) \rightarrow \text{Path}_A(M,N)
\]

\[
\text{coerce } x \rightarrow 0 \rightarrow 1 \quad \text{in Path}_A(M,N) \rightarrow \text{Path}_A(M,N)
\]

If you have an \(x\)-line, it ought to go from the left endpoint to the right endpoint.

**Coercion along a type line**

**Intuition:** the line will give you everything you need to do the coercion.

Wants a generic notion of coercion

\[
\text{coerce } x \rightarrow 0 \rightarrow 1 \quad \text{in } A \langle 0,1 \rangle \rightarrow A \langle 0,1 \rangle
\]

(this will end up being an equivalence,

given \(A\) a type \([I,z]\)

\[
\text{coerce } x \rightarrow 0 \rightarrow 1 \quad \text{in } A \langle 0,1 \rangle \rightarrow A \langle 0,1 \rangle
\]

\[
\text{coerce } x \rightarrow 0 \rightarrow 1 \quad \text{in } A \langle 0,1 \rangle \rightarrow A \langle 0,1 \rangle
\]

This is critical

**How could this work:** `coerce` looks at \(A\) to know what to do, and \(A\) tells you how you coerce. `coerce` unpacks the equiv given by \(A\), applies it to the left end and gets the right end.

**Cases of coercion**

**Easiest** `
\[
\text{coerce } x \rightarrow 0 \rightarrow 1 \quad \text{in } A \langle 0,1 \rangle
\]

\[
\text{coerce } x \rightarrow 0 \rightarrow 1 \quad \text{in } A \langle 0,1 \rangle
\]

**Easy** `
\[
\text{coerce } x \rightarrow 0 \rightarrow 1 \quad \text{in } A \langle 0,1 \rangle \rightarrow A \langle 1,2 \rangle \quad \text{depends on } A
\]

\[
\text{coerce } x \rightarrow 0 \rightarrow 1 \quad \text{in } A \langle 1,2 \rangle \rightarrow A \langle 0,2 \rangle
\]
This is an x-line!

\[ \text{Coe} \circ \vartheta_{x,A} \circ \vartheta_{x,A}^{-1} = \text{Coe} \circ \vartheta_{x,A}^{-1} \circ \vartheta_{x,A} = \text{id} \]

\[ \text{Coe} \circ \vartheta_{x,A}^{-1} \circ \vartheta_{x,A} = \text{Coe} \circ \vartheta_{x,A}^{-1} \circ \vartheta_{x,A} \circ \text{id} = \text{id} \]

\[ \text{Coe} \circ \vartheta_{x,A}^{-1} \circ \vartheta_{x,A} \circ \text{id} = \text{id} \]

This line is the x-line connecting \( M_0 \) to \( M \) and \( M_0 \) to \( M \) in the heterogeneous line. The lines are classified by line.

\[ M_0 \xrightarrow{\text{Coe} \circ \vartheta_{x,A}^{-1}} \text{Coe} \circ \vartheta_{x,A}^{-1}(M_0) \]

\[ \text{A} \xrightarrow{M} \text{A} \]

\[ \text{A}_0 \xrightarrow{M} \text{A} \]

New time: \( \text{Coe} \circ \vartheta_{x,A} \) ends up being a square.
Last time

- Cubical "pre-type" theory
  Types & elements are structured as Cartesian cubes of arbitrary dimension

\[
\begin{align*}
A & \Rightarrow \text{type } [x] \\
M & \Rightarrow \text{type } [y]
\end{align*}
\]

- NB: type judgments work naturally (uniform with dimension)
  \[
a : A \Rightarrow N(x) \Rightarrow [y]
\]
  \[
\text{iff } \forall y : T \rightarrow T \text{ if } M = M' \epsilon A \text{ then } \text{N}(x) = \text{N}(M'/y) \in [M/y]B
\]

- Mental picture to keep in mind (the cubes draw it upside down)

---

Type lines are what matter!
(Element lines are there to induce type lines)

\[
a : A \Rightarrow F \text{ type } [x, y]
\]

Sends \(x\)-lines in \(A\) to type lines
If \(M \in A \Rightarrow [x, y]\), \(F(M, a) \Rightarrow \text{type } [a, x]\)

An important "other" type line is given by an equivalence (ise up to path). Approx: if \(A \equiv B\) induces \(u\_A(E) \Rightarrow \text{type } [x, y]\)
Such that \(u\_A(E) = A \& u\_B(E) = B\)

(126)
To be a type (as opposed to a pre-type) requires additional conditions called the Kan conditions.

1. Coercion along a type line

\[ \text{coe}^\text{pre} : A \rightarrow A' \rightarrow A \]

**Idea:** think of \( A \) as induced by \( F \)

\[
\begin{align*}
\frac{\text{F}\{H\}}{\text{H} \cdot \text{T}} \quad \text{"F[\text{E}]"} \quad \quad \quad \text{"F[\text{E}]"} \\
\text{coe represents the transport in HoTT} \quad \text{\text{F}[\text{E}]\text{ represents the transport in HoTT}}
\end{align*}
\]

does so in 2 stages:

1. \( \text{F}[\text{E}] \) induces a type line
2. The coercion activates (runs it as a program) the type line

\[ \text{coe}^\text{cmp} : A(0/\times) \rightarrow A(1/\times) \]

\[ \text{coe}[A.F](p) \]

We examined various coe's.

There's a funny, subtle interplay between type oblivious computation & type awareness.

\[ \begin{align*}
\text{coe}^\text{com} : A(M) \rightarrow M & \quad \text{type-indep} \\
\text{coe}^\text{cmp} : (M) \rightarrow M & \quad \text{dependo} \\
\text{coe}^\text{cmp} : (M, \text{coe}^\text{com} (M)) & \quad \text{or A!} \\
\text{coe}^\text{cmp} : (M \cdot A(0/\times)) & \rightarrow A(1/\times) \quad [x,y]
\end{align*} \]

these are both y-lines

Remark, they could already have varied in \( y \).

\[ \begin{align*}
\text{coe}^\text{com} : A(0/y) & \rightarrow A(0/y) \rightarrow A(0/y) \rightarrow A(0/y) \rightarrow A(0/y) \\
\text{coe}^\text{cmp} : M(0/y) & \rightarrow A(0/y) \rightarrow A(0/y) \rightarrow A(0/y) \rightarrow A(0/y) \\
\text{coe}^\text{cmp} : (M(0/y), A(0/y)) & \rightarrow A(1/y) \rightarrow A(1/y) \rightarrow A(1/y) \rightarrow A(1/y)
\end{align*} \]

- \text{dependent on A}(1/y), could be computable
due to taking a face. This is why you need coherence; taking face commutes in computation.

$\text{coe}$ $^{\text{sym}}_x A$ — symmetric to previous

$\text{coe}$ $^{\text{sym}}_x A$ (M) $\Rightarrow$ 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 like you

need to worry about coherence.

Suppose $y \equiv y'$. Then the only possible

reduction of

$\text{coe}$ $^{\text{sym}}_x A$ (M) $\Rightarrow$ M'

are determined by $A$ $\Rightarrow$ (gillion steps)

Now suppose you take

$\text{coe}$ $^{\text{sym}}_x A$ (M) $\langle y/y' \rangle$ $\Rightarrow$ $\text{coe}$ $^{\text{sym}}_x A$ (M $\langle y/y' \rangle$) $\Rightarrow$ M $\langle y/y' \rangle$ $\Rightarrow$ x

and you need to make sure you get

the same thing.

Given $M \in A [I, x]$,

$M_0 \xrightarrow{M} M_1 \in A \xrightarrow{A} A, [I, x]$

The Master

Diagram
Type-specific coercion \( \text{coe}_{x:A} (M) \) (\( x + A \))

Role is closed, so it doesn't depend on the role. Let's figure out what this should be:

\[
\text{coe}_{x:A} (M) \rightarrow M
\]

No variation \( \rightarrow \) nothing to do.

Now suppose we have variation:

\[
\text{coe}_{x:A} (M : A < x/2 \rightarrow B < n/x>) \rightarrow A < x/2 > B < n/x>
\]

What makes this correct? It requires a story, which we'll get into later.

\[
\text{coe}_{x:A} (M : A < x/2 \rightarrow B a < x/2 >) \rightarrow A < x/2 > B < a/2 >)
\]

\[
\lambda x : A < x/2 >.
\]

We're kind of stuck. We want:

\[
B < n/x > [a/a]
\]

for \( e : B \)...

Is there any way to get

\[
B < n/x > [\text{coe}_{x:A} (c) / c] \rightarrow B < n/x > [a/a]
\]

Build a tree & rely on the action of \( B \) on lines!
\[ B(x/x) \{ \text{coe}^{\times_{E}}_{A} (a)/a \} \]

Then you get the answer:

\[ A = A(x' / x) \]

\[ \text{coe}^{\times_{E}}_{A} \left[ \text{coe}^{\times_{E}}_{A} (a) / a \right] \]

\[ (M \circ \text{coe}^{\times_{E}}_{A} (a)) \]

\[ \begin{array}{c}
\text{x-line} \\
B(x' / x) \{ \text{coe}^{\times_{E}}_{A} (a)/a \}
\end{array} \]

\[ B(x' / x) \{ a/a \} \]

**Exercises:**

1. \[ \text{coe}^{\times_{E}}_{A} (M \in \text{A}(x,B,x')) \in (A \times_{X} x) \times_{x} B \times_{X} x' \]

2. \[ \text{coe}^{\times_{E}}_{A} (M \in a: A \times_{X} x \times B \times_{X} x') \in a: A \times_{X} x \times B \times_{X} x' \]

Just carefully follow your nose.

**Consider now**

\[ \text{coe}^{\times_{E}}_{A} \left( \text{R} \in \text{R}(x) \right) \]

\[ \text{coe}^{\times_{E}}_{R} \left( \text{P} \in \text{P}(x) \right) \]

\[ \text{P} \in \text{P}(x) \]

\[ \text{P} \in \text{P}(x) \]

Recall, there are quoted paths of the form \(< y, y \)>

This is non-trivial & forces us to consider another condition on paths called Ken composition.
Then we will get that a

\[ \text{type} = \text{pre-type} \\
+ \text{Kan coercion} \\
+ \text{Kan composite} \text{ (enough line)} \]
Given $\text{MC}\{F, x\}$

\[
\begin{array}{c}
\text{M}_0 \xrightarrow{M} \text{M}_1 \in A_0 \xrightarrow{A} A_1
\end{array}
\]

we have the square

\[
\begin{array}{c}
\text{M}_0 \xrightarrow{\text{coe}_x F (\text{M}_0)} \text{coe}_x \text{A}_x (\text{M}_0) \\
\downarrow \text{coe}_x \text{A}_x (\text{M}_0) \quad \downarrow \text{coe}_x \text{A}_x (\text{M}_1) \quad \text{coe}_x \text{A}_x (\text{M}_1) \in A_y
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x \text{A}_x (\text{M}_0) \xrightarrow{x \cdot A} \text{M}_1 \in A_1
\end{array}
\]

(In case $x \cdot A_x (\text{M}_0)$, the $x$ in $x \cdot A_x$ is free, and the $\forall x$ is bound)

For all types $A$, $\text{coe}_x: A \rightarrow M$. The other cases depend on $A$.

$\text{coe}_x: A \rightarrow M$

\[
\begin{array}{c}
\text{coe}_x F (\text{M}) \rightarrow B<\text{f}/x>
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A \rightarrow B (\text{M}) \in A <\text{f}/x> \xrightarrow{\text{coe}_x A \rightarrow B (\text{M})} B<\text{f}/x>
\end{array}
\]

$\lambda e A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))$

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]

\[
\begin{array}{c}
\text{coe}_x A<\text{f}/x>, \text{coe}_x A \rightarrow B (\text{M} (\text{coe}_x A (a)))
\end{array}
\]
Given $M \in \text{Path}_{x:A \otimes y}((x, y), (p_0, p_1))$, we know $M @ 0 = p_0 \triangleq p_1 <(x, y) > \in A^{r^{(x, y)}}$.

Want

$\{ ? \}
\begin{align*}
\forall x, y, A^{r^{(x, y)}} \subseteq A^{r^{(x, y)}}[x, y] \\
\forall p_0, p_1, p_{<x, y>}, p_{<x, y>}', A^{r^{(x, y)}} \subseteq A^{r^{(x, y)}}[x, y] \\
\end{align*}$

Then $<x, y> \in \text{Path}_{x:A \otimes y}((x, y), (p_0 \triangleq p_1, p_0 \triangleq p_1))$, which is what we want.

What if we took the line

$\forall x, A^{r^{(x, y)}} \subseteq A^{r^{(x, y)}}[x, y]$,

We know that $p_0 \in A^{r^{(x, y)}}[x, y]$, so $p_0 \triangleq p_1 <(x, y) > \in A^{r^{(x, y)}}$.

The endpoints are in the right type, but the endpoints aren't the ones we want ($p_0 <(x, y) >$ vs. $\forall x, A^{r^{(x, y)}}[x, y]$).

Note: $\forall x, A^{r^{(x, y)}} \subseteq A^{r^{(x, y)}}[x, y]$

1. $\forall x, A^{r^{(x, y)}} \subseteq A^{r^{(x, y)}}[x, y]$
2. $\forall x, A^{r^{(x, y)}} \subseteq A^{r^{(x, y)}}[x, y]$.

When $y = r$, we get

1. $\forall x, A^{r^{(x, y)}} \subseteq A^{r^{(x, y)}}[x, y]$
2. $\forall x, A^{r^{(x, y)}} \subseteq A^{r^{(x, y)}}[x, y]$.
If lines are data witnessing the interchangeability of ends, it stands to reason that we should be able to interchange the \( P_i < r' i \). To get this, we require our types to have/contain Kan composition, which we introduce now:

\[
? = \text{hcom}_{A< \alpha_y> }^{\text{mor} \ (M@z)} \text{COE}_{y,A< \alpha_y>} (M@z) \cdot \begin{cases} 
  x = 0 & \rightarrow y . \text{COE}_{y,A< \alpha_y>} \ (P_0) \\
  x = 1 & \rightarrow y . \text{COE}_{y,A< \alpha_y>} \ (P_1) 
\end{cases}
\]

Cleaner notation: "\( \text{COE}^{\text{mor} \ (M@z)}_{y,A} \)"

and then the image of \( M \) becomes the quotient of this composition.

\( \text{COE} = \) what do you do with the lines you have, \( \text{hcom} = \) you have enough lines.
Topic: Kan conditions on pre-types sufficient to be types.

Pretypes — cubical structure
- ramifications into lim
- higher claims are supposed to presuppose identification (towards for interchangeability)
  "Why?" 1. coercion along a type identification
  \[ \text{coe}_{\mathcal{Y}}(M) \in A \langle \mathcal{X} \rangle \]
  "What?" 2. composition of identifications
  \[ \text{hcom}_{\mathcal{A}}^\mathcal{Y}(M, x : y \rightarrow z, y) \]

A motivation for composition:
derive coe in a Path type.

\[ \text{coe}_{\mathcal{Y}}(A \langle x \rangle) \rightarrow (P_0 \langle x, y \rangle) \rightarrow (M \langle x \rangle) \rightarrow (P_1 \langle x, y \rangle) \]

\[ \text{coe}_{\mathcal{Y}}(A \langle x \rangle) \rightarrow \text{coe}_{\mathcal{Y}}(M \langle x \rangle) \rightarrow \text{coe}_{\mathcal{Y}}(P_1 \langle x, y \rangle) \]

The identifications we want is noted, thus:

The diagram above:

\[ \text{hcom}_{\mathcal{A}}(\mathcal{X}, \text{coe}_{\mathcal{Y}}(M \langle x \rangle), \text{coe}_{\mathcal{Y}}(P_0 \langle x \rangle)) \]

\[ \text{hcom}_{\mathcal{A}}(\mathcal{X}, \text{coe}_{\mathcal{Y}}(M \langle x \rangle), \text{coe}_{\mathcal{Y}}(P_1 \langle x \rangle)) \]

homogeneous (invariant, implicit adjacency, contains, this when \( x = 0 \))

aka \( \text{com}_{\mathcal{A}}(M \langle x \rangle; P_0 \langle x \rangle) \) when \( x = 1 \)

Derived form:

\[ \text{com}_{\mathcal{A}}(M \langle x \rangle; P_0 \langle x \rangle) \]

We claim this is the general form/case.

eg) \( a \rightarrow b \) is a ! . \( b \)

Transitivity:

\[ P_b \rightarrow P_a \]

(135)
Symmetry, aka reversal, inversion.

(Can derive higher identifications witnessing groupoid laws

\( p \cdot (q \cdot r) \)

\[ \begin{array}{c}
\text{line between lines} \\
(1) \quad (2)
\end{array} \]

The validity of such an identification is due to a higher identity, 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 for finite beings can speak of the infinite is via computation)

Kan conditions (to be a type)

A type \([F]\) means

1) \( A \xrightarrow{f} A_0 \) and \( A_0 \) is a canonical type in \( F \)
2) coercion structure

\( f : F \Rightarrow F' \), \( x \rightarrow \) then for all \( x, x' \)

\( \text{coer}_{x, x'}^\mu \in A < x, x' > \rightarrow A < x, x' > \)

\( \text{coer}_{x, x'}^\mu : \text{id}_{A < x, x '>} \rightarrow F \)
3) Composition Structure

The following exist:

1-D line

2-D filler

"composite" "filler" which fills forms the square

hecom \( y_0 \) hecom \( y_{y_0} \)?

Special case

Given \( M \in \text{A}[\mathcal{F}, x] \) "cap"

\( N_0 \in \text{A}[\mathcal{F}, x, y_0 \mid x = 0] \) "tube"

\( N_1 \in \text{A}[\mathcal{F}, x, y_1 \mid x = 1] \) "tube"

The \( N_0, y_0 \dot{=} M \in \text{A}[\mathcal{F}, x, y \mid x = 0] \)

we obtain \( N, x, y \dot{=} \in \text{A}[\mathcal{F}, x, y \mid x = 1] \)

we obtain \( \text{hecom}_x (M \{ x \Rightarrow y, y, M, y \}) \in \text{A}[\mathcal{F}, x] \)

if \( r = r' \), then \( \text{hecom}(\#) = M \in \text{A}[\mathcal{F}, x] \)

if \( r \not= r' \), then \( \text{hecom}(\#) = N_0, x, y \in \text{A}[\mathcal{F}, x, y \mid x = 0] \)

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?

\( x \)

\( y \)

\( z \)

[Diagram of a cube with labels]

Top, bottom, L, R bases = tube

Entire cube = filler

You have 2 sets of adjacency constraints:
1) the tube face must be adjacent to the cap

2) the tube face 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: e.g. Start with B & Top, Bottom fill in sides then get front vs. Start out in the 7 faces & then do the front.

What about diagonals?

They are a way of expressing coherence!!! That will be the key to making univalence work.

General Case

This is a theorem about the semantics.

Cliff A type \([F]\)

\(M : A \to [F]\)

\(\Pi_i = \Pi'_i\) valid

(Cap adj)

\((\forall i) N_i : A \to [F] \mid 2_i = 2'_i\)

(Tube adj)

\((\forall i j) N_i = N_j \in A \to [F] \mid 2_i = 2'_i, x_j = x'_j\)

Then theorem \(\Pi_i, (M : A \to [F]) \vdash A [F]\)

With if \(2 = 2'\), then \(\Pi_i, (M : A \to [F])\)

\(\vdash A [F] \to A [F] \)
$r_i = r_i''$ valid:

either

1) $r_i$ is $r_i'' (E_i)$

   $0 = 0, 1 = 1, x_i = x_i''$

or

2) $r_i = r_j, r_i'' = 0, r_j'' = 1 (E_i, j)$

the same

\[
\begin{array}{c|c|c}
\hline
x = 0 & x = 1 \\
\hline
i & j \\
\hline
\end{array}
\]

What are the composition rules?

$hcomp_{\mathcal{A}} (M, -) \rightarrow M$

$hcomp_{\mathcal{A}} (M; \exists \bar{e} \rightarrow y. N_i, -)$

$\rightarrow N_i < r_i', /y>$ want a deterministic op. sem.

$hcomp_{\mathcal{A}} (M; \exists \bar{e} \rightarrow y. N_i)$

$\rightarrow hcomp_{\mathcal{A}} (M; \exists \bar{e} \rightarrow y. N_i) \text{ val}$

"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 books.)
Last time: Kan conditions

(Whaa) 1. Coercion \( \alpha : A \xrightarrow{x} A \xrightarrow{\alpha /x} A \alpha /x \)

(Which) 2. Composition \( \hom \alpha \quad (M; \xi \xrightarrow{\alpha } y, N) \)

\[ \xi \text{ are are slim equations } \Rightarrow \text{ some is true or there is a pair } r=0, r=1. \]

\( \hom \in A[\xi] \text{ when } \)
\[
M \in A[\xi] \quad \text{adjacency}
\]
\[
N \in A[\xi, y] \quad \text{effective adjunctions}
\]

\[
\text{Visually, it helps to think of the cube and (}
\]

( the 0, 1 cases)

Defining \( \hom \in A \) for various \( A \)

1) strict booleans ("built-in" by hand)

\( \text{Bool} \) — observables for canonicity:
\( \text{closed terms should evaluate to true or false. Imply termination.} \)

\( \text{Bool val} \quad \text{Bool} = \text{Bool type } \quad [\xi] \)
\( \text{tt, ff val} \quad \text{tt} = \text{tt} \in \text{Bool } \quad [\xi]
\)
\( \text{ff} = \text{ff} \in \text{Bool } \quad [\xi] \)

\[
\begin{align*}
& \forall x, A (H; M_0, M_1) \rightarrow y. A (H; M_0, M_1) \quad \text{if } H \rightarrow M_1 \\
& \forall (tt; H_0; M_1) \rightarrow M_0 \\
& \forall (ff; H_0; M_1) \rightarrow M
\end{align*}
\]

We claim this is a Kan type, so we must
gives composition.
\[ \text{Coe} \circ \text{?} \quad (M) \to M \]
\[ \text{loc} \circ \text{?} \quad (H, \varepsilon_i \to y, N_i) \to M \]

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) \text{if } \text{Bool} \text{ weak Booleans ("incl. defined")}

free

the \text{Kan} type generated by \text{true} \& \text{false} \text{ generators: } \text{true}, \text{false} \in \text{Bool} \[ \text{val} \]

For technical reasons, last time we had
\text{loc} \circ \text{?} \text{ from } \text{val}. \text{We'll just do}
for today, the following

\[ \text{loc} \circ \text{?} \quad (M; \varepsilon_i \to y, N_i) \text{ val } \]

\[ \alpha \text{[ ]} \text{ [ ]} \text{ H } \quad \text{ We are oblivious of the } \]

\[ \alpha \text{[ ]} \text{ [ ]} \text{ ] } \text{ fact that adjoining } \]

\[ \alpha \text{[ ]} \text{ [ ]} \text{ ] } \text{ squares to be unremarkable, instance } \]

\[ \alpha \text{[ ]} \text{ [ ]} \text{ ] } \text{ of } \text{tttt} \text{ or } \text{ffff}. \text{ So we } \]

\[ \alpha \text{[ ]} \text{ [ ]} \text{ ] } \text{ just throw in the } \text{loc} \text{ as a } \text{pseud } \]

\[ \alpha \text{[ ]} \text{ [ ]} \text{ ] } \text{ object as call it a day } \]

\[ \alpha \text{[ ]} \text{ [ ]} \text{ ] } \text{ there are infinitely many Booleans! } \]

\[ \beta \alpha \text{[ ]} \text{ [ ]} \text{ A[A[ttt]] [ttt] } \text{ "Motiv" } \]

\[ \beta \alpha \text{[ ]} \text{ [ ]} \text{ ] } \text{ "property of } M \]

\[ \beta \alpha \text{[ ]} \text{ [ ]} \text{ ] } \text{ provided } \{ \text{Mo } \in \text{A[ttt]} \}

\[ \alpha \text{[ ]} \text{ [ ]} \text{ ] } \text{ "Type } \]

\[ \alpha \text{[ ]} \text{ [ ]} \text{ ] } \text{ provided } \{ \text{Mo } \in \text{A[ttt]} \}

\[ \alpha \text{[ ]} \text{ [ ]} \text{ ] } \text{ property of } M \]
But what do you do for

If $A (\hom \rightarrow (M; \Sigma \rightarrow yN); \text{Book}) \rightarrow \square$

$\text{Book: } (F)$

The $\hom$ is an $x$-line. This feels a lot like the embarrassing situation we were in where we had inhabitants of an identity type that weren't left.

What to do?

If $A (-, M_0, M_1) \rightarrow F(-)$

Want $b : \text{Book} \Rightarrow F(b) \in A[b/a] [E]$.

What do we know about these mappings?
We have a presheaf semantics!
The mapping $\rightarrow$ sends points to points
$\rightarrow$ sends lines to lines
$\Rightarrow$ preserves identifications!

If $A (\hom (-, M_0, M_1)) \rightarrow ?$

Want to preserve the $\hom$ line.
What does this mean?
"$F \in \text{Book} \rightarrow A"$

Can assume $A$ is Kan, because our whole setup assumes we live in a world where all types are Kan

A hom composition!
Take the line in Book and ask $A$ to make a line out of it.
Need to know the motive!
What should $\otimes$ be?

Let's look at the $z$ line:

1. $\text{Hom}_{\text{N}}(M; \xi \rightarrow y, M)$

$z_{\text{line}} \otimes x = M$

$z_{\text{line}} \otimes x(t_2) = \text{Hom}_{\text{N}}(M, \xi \rightarrow y, M)$

So we have a $z$ line

$M \rightarrow \text{Hom}_{\text{N}}(M, \xi \rightarrow y, M)$

Need to show $\text{F}(M) \in \text{A}[\text{Hom}_{\text{N}}(\xi \rightarrow y, M)] = \text{A}[\text{N}]$

$\text{F}(N; \circledast) \in \text{A}[\text{Hom}_{\text{N}}(\xi \rightarrow y, M)] = \text{A}[\text{N}(\circledast)]$

Because the type varies in $\xi$, we can't use $\otimes$

Rather, we must generalize to a $\otimes$:

**Heterogeneous Composition**

$\text{Hom}_{\text{N}}(M; \xi \rightarrow y, M)$

$\rightarrow \text{A}[\text{Hom}_{\text{N}}(M; \xi \rightarrow y, M)]$

$\rightarrow (\text{F}(M; \xi \rightarrow y, \text{F}(N; \circledast)))$

So we get:

$\text{F}(\text{Hom}_{\text{N}}(M; \xi \rightarrow y, M)) \rightarrow$

$\rightarrow \text{Hom}_{\text{N}}(\text{F}(M; \xi \rightarrow y, \text{F}(N; \circledast)))$

So, in short, we know what to do on $\xi, N$, and when we see an $\text{Hom}$, 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) $A \rightarrow B$ (no dep.)

- $\text{hom}_{A \rightarrow B}(M; \xi \rightarrow y.N_i)$
- $\rightarrow a \cdot \text{hom}_{B}(M(a); \xi \rightarrow y.N_i(a))$

Adjacency still holds because if the two functors are adj, then they remain adj when applied.

$$\begin{array}{ccc}
N_0 & \rightarrow & N_1 \\
\downarrow & & \downarrow \\
N_i(a) & \rightarrow & N_i(a)
\end{array}$$

4) $a \cdot A \rightarrow B(a)$

- $\text{hom}_{a \cdot A \rightarrow B(a)}(M; \xi \rightarrow y.N_i)$
- $\rightarrow a \cdot \text{hom}_{B(a)}(M(a); \xi \rightarrow y.N_i(a))$

5) $a \cdot A \times B(a)$

Need to be careful: depending how you write.

$\text{hom}_{a \cdot A \times B(a)}(M; \xi \rightarrow y.N_i)$

To dep.

$$\begin{array}{ccc}
\text{hom}_A (M; \xi \rightarrow y.\text{ftr}(N_i)) \\
\downarrow & & \downarrow \\
b \cdot \text{hom}_B(a; \xi \rightarrow y.\text{ftr}(N_i))
\end{array}$$

$$\begin{array}{ccc}
\rightarrow \text{hom}_{A \times B(a)}(M; \xi \rightarrow y.\text{ftr}(N_i)) \\
\downarrow & & \downarrow \\
\text{hom}_{A \times B(a)}(M; \xi \rightarrow y.\text{ftr}(N_i))
\end{array}$$
6) Check for yourself: Path \( x \rightarrow P \supset (P, Q) \)

\[ \text{com}_{\text{Path}_{x, A}(P; A)} (M; \xi_i \rightarrow y, N \xi_i) \]

Hint: Dimension shifts augment constraints \( x = 0, x = 1 \).