Martin-Löf's Type Theory



From $\lambda \rightarrow \Sigma +$ to Type Theory

We have:

- Dependent types, encodes FOL up to cut-elimination for logical cuts (for \land , \lor , \Rightarrow , \forall , \exists)
- System T: allows the definition of complex computations $R_T: T \to (N \to T \to T) \to N \to T$

Let us add the induction axiom :

 $R_P: (P \ 0) \rightarrow (\Pi \ x : N \ (P \ n) \rightarrow (P \ (S \ n))) \rightarrow \Pi \ n : N \ (P \ n)$

 $(R_P p_0 p_S 0) \triangleright p_0$ $(R_P p_0 p_S (S n)) \triangleright p_S n (R_P p_0 p_S n)$







Subject reduction

$(R_P p_0 p_S 0) \triangleright p_0$

 $R_P: (P \ 0) \rightarrow (\Pi \ x : N \ (P \ n) \rightarrow (P \ (S \ n))) \rightarrow \Pi \ n : N \ (P \ n)$

```
Suppose (R_P p_0 p_S 0): T
so p_0: (P 0)
SO p<sub>S</sub>: (\Pi x : N . (P n) \rightarrow (P (S n)))
so: 0 : N (it is)
so: T =_{\beta} P 0
so: p0 : T
```

Obviously similar for $(R_P p_0 p_S (S n)) \triangleright p_S n (R_P p_0 p_S n)$





Equality and equality cuts

 $I_T: T \rightarrow T \rightarrow Type$ (provided T : Type) $\operatorname{refl}_{\mathrm{T}}: \Pi \mathbf{x} : \mathrm{T} \cdot \mathrm{I}_{\mathrm{T}} \mathbf{x} \mathbf{x}$ $L_P : \Pi x : T. \Pi y : T. P x \rightarrow I_P x y \rightarrow P y$

 $L_P a b p (refl_T c) \triangleright p$

 $L_P a b p (refl_T c) : Q$ so: $P: T \to Type$, a: T, b: T, p: P a, $Q =_{\beta} P b$ since $(\operatorname{refl}_T c)$: $I_T c c$, we have $I_T c c =_{\beta} I_T a b$, so $a =_{\beta} b =_{\beta} c$ so by conversion, p : P b



formally: $\Gamma \vdash T : Type$ $\Gamma \vdash I_T: T \to T \to Type$



A last detail

We want to talk about falsity

Γwf Γ⊢⊥:Type

 $\frac{\Gamma \vdash t: \bot \quad \Gamma \vdash T:Type}{\Gamma \vdash efq_T(t):T}$

Nothing more to be added



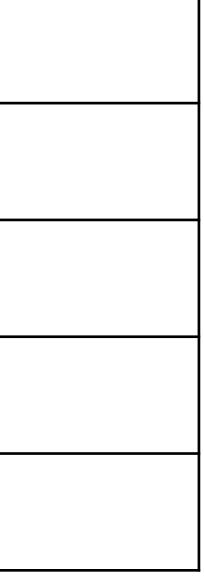


Normalization

We can still map normalization to the corresponding system without dependent types. That is System T with product and sum types

N	Ν
I _T tu	Ν
Σ x:A.B	A×B
A+B	A+B
П x:A.B	$A {\rightarrow} B$





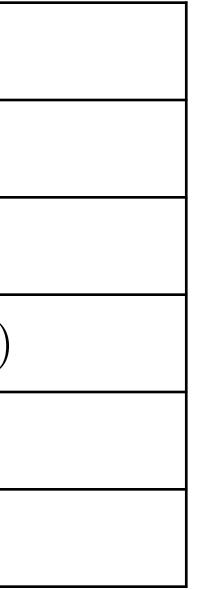


Closed normal terms

A closed normal term of type ... is of the form(s) ...

N	0, S(t)
I _T t u	$\mathrm{refl_T} \ \mathrm{v}$
Σ x:A.B	(a, b)
A+B	i(a), j(b)
П x:A.B	λ x:A'.t
	none







By normalizing a proof of $\Sigma x:A.B$, we obtain t:A and $p:B|x \setminus t|$ By normalizing a proof of A+B, we obtain either i(A) or j(b)From a proof $t: \Pi x:A$. $\Sigma y:B.C$ we can obtain: $f: A \rightarrow B$ $p:\Pi x:A. C|y \setminus (f x)|$ (furthermore f is typable in System T)

There is no term t s.t. [A:Type] \vdash t : A + (A $\rightarrow \perp$) There are closed A:Type with no term t s.t. [A:Type] \vdash t : A + (A $\rightarrow \perp$) (and other variants)



$f = \lambda x: A \cdot \pi_1(t x)$ $p = \lambda x: A \cdot \pi_2(t x)$



Suppose we have $[] \vdash p : I_N 0 (S 0) \rightarrow \bot$ by erasing type dependencies, we get: $[] \vdash |p| : N \rightarrow \bot$ and thus a closed term of type \perp (in System T or MLTT) Hence: $0 \neq 1$ is not provable in MLTT.

Indeed having a discrimination predicate means having "really dependent types": EQZ 0 > T EQZ (S_) $\triangleright \perp$

What happens in Coq?







nat rec : Π P : nat \rightarrow Type. P 0 \rightarrow $(\Pi m : nat. P m \rightarrow P (S m)) \rightarrow$ Π n:nat. P n

nat rect : Π P : nat \rightarrow Kind. P 0 \rightarrow $(\Pi m : nat. P m \rightarrow P (S m)) \rightarrow$ Π n:nat. P n

We can then define $EQZ = (nat_rect \lambda x:nat.Type)$ True λ m:nat. λ X:Type. False)





Modern Coq

Definition EQZ (n:nat) : Type := match n with $0 \Rightarrow True$ S => False end.

In Coq, operators like R_T are not primitive, but built by combining:

- pattern-matching
- structural recursion





Universes

In Coq: Type₁ : Type₂ : Type₃ : ... together with pattern-matching (R operators) towards all Type

It is interesting to look at the restrictions over eliminations of inductive types to keep the system consistent (but done in 2-7-2?)

with: Martin-Löf's proposal: An addition type U: Type (universe) "constructors" for this type: tr n > natn:U tr bot $\triangleright \perp$ bot : U $\pi: \Pi a: U. (tr a) \rightarrow U) \rightarrow U$



$tr: U \to Type$ tr $(\pi a f) \rightarrow \Pi$ x:tr a. tr(f x)

Type Univ	erses in Martin-Lö
Martin-Löf's proposal:	with:
An addition type U: Type ((universe)
"constructors" for this type:	$\mathrm{tr}:\mathrm{U}\rightarrow$
n:U	$\operatorname{tr} n \triangleright n$
bot : U	$\mathrm{tr}\ \mathrm{bot}\ \triangleright$
$\pi:\Pi a{:}U. (tr a){\rightarrow} U) \rightarrow U$	
• • •	$tr(\pi a f)$

U is an inductive type which allows to model all types (except U) tr is defined by pattern-matching + recursion but tr occurs in the definition of U: so-called inductive-recursive type (this last feature is not available in Coq)

> What does a universe hierarchy look like in this setting? Which extension would be paradoxical?

of style



- → Type nat
- > _
- $f) \rightarrow \Pi x$:tr a. tr(f x)



Coq Exercise

Construct div2 using only:

- Definition
- nat_rec (or match with + Fixpoint)

```
Definition P2 n :=
         {p: nat & {n = p + p}+{ n = S (p + p)}}.
div2 (n : nat) : P2 n
```

