Martin-Löf's Type Theory

From λ→Σ+ to Type Theory

We have:

- Dependent types, encodes FOL up to cut–elimination for logical cuts (for ∧, ∨, ⇒, ∀, ∃)
- 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 :

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

$$(R_P\ \ p_0\ p_S\ \ 0) \quad \triangleright \quad p_0$$

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

Subject reduction

$$(R_P\ \ p_0\ p_S\ \ 0)\quad \triangleright \quad p_0$$

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

Suppose $(R_P\ p_0\ p_S\ \ 0) : T$

so p_0: $(P\ 0)$

so p_S: $(\Pi\ x : N\ .\ (P\ n) \to (P\ (S\ n)))$

so: $0 : N$ (it is)

so: $T =_\beta P\ 0$

so: $p0 : T$

Obviously similar for

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

Equality and equality cuts

$I_T : \ T \to T \to \mathrm{Type}$ (provided T : Type)

$\mathrm{refl}_T : \Pi\, x : T\ .\ I_T\ x\ x$

$L_P : \Pi\, x : T.\ \Pi\, y : T.\ P\ x \to I_P\ x\ y \to P\ y$

formally:

$$\frac{\Gamma \vdash\ T : \mathrm{Type}}{\Gamma \vdash\ I_T : T \to T \to \mathrm{Type}}$$

$$L_P\ a\ b\ p\ (\mathrm{refl}_T\ c) \ \triangleright\ p$$

$L_P\ a\ b\ p\ (\mathrm{refl}_T\ c) : Q$

so : $P : T \to \mathrm{Type}, \quad a : T, \quad b : T, \quad p : P\ a, \quad Q =_\beta P\ b$

since $(\mathrm{refl}_T\ c)$: $I_T\ c\ c$, we have $I_T\ c\ c =_\beta I_T\ a\ b$, so $a =_\beta b =_\beta c$

so by conversion, $p : P\ b$

A last detail

We want to talk about falsity

$$\frac{\Gamma \text{ wf}}{\Gamma \vdash \bot : \text{Type}} \qquad \frac{\Gamma \vdash t : \bot \quad \Gamma \vdash T : \text{Type}}{\Gamma \vdash \text{efq}_T(t) : T}$$

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	N
$\mathrm{I_T}\ t\ u$	N
$\Sigma\ x{:}A.B$	$A \times B$
$A+B$	$A+B$
$\Pi\ x{:}A.B$	$A \to B$

Closed normal terms

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

N	0, S(t)
I_T t u	refl_T v
Σ x:A.B	(a, b)
A+B	i(a), j(b)
Π x:A.B	λ x:A'.t
⊥	none

The constructive paradise

By normalizing a proof of $\Sigma\ x{:}A.B$, we obtain $t{:}A$ and $p : B[x\backslash\ t]$

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:

$f : A \to B$

$p : \Pi\ x{:}A.\ C[y \setminus (f\ x)]$

$f \equiv \lambda\ x{:}A\ .\ \pi_1(t\ x)$

$p \equiv \lambda\ x{:}A\ .\ \pi_2(t\ x)$

(furthermore f is typable in System T)

There is no term t s.t. $[A{:}\mathrm{Type}] \vdash t : A + (A \to \bot)$

There are closed $A{:}\mathrm{Type}$ with no term t s.t. $[A{:}\mathrm{Type}] \vdash t : A + (A \to \bot)$

(and other variants)

Limits of this Type Theory

Suppose we have $[] \vdash p : I_N\ 0\ (S\ 0) \to \bot$

by erasing type dependencies, we get: $[] \vdash |p| : N \to \bot$

and thus a closed term of type ⊥ (in System T or MLTT)

Hence: 0≠1 is not provable in MLTT.

Indeed having a discrimination predicate means having "really dependent types" :

$\text{EQZ}\ 0 \triangleright \top$

$\text{EQZ}\ (S\ \_) \triangleright \bot$

What happens in Coq ?

Historical Coq

```
nat_rec : Π P : nat→ Type. P 0 →
                   (Π m : nat. P m → P (S m)) →
                     Π n:nat. P n

nat_rect : Π P : nat→ Kind. P 0 →
                   (Π m : nat. P m → P (S m)) →
                     Π n:nat. P n
```

We can then define

EQZ ≡ (nat_rect λ x:nat.Type
                    True
                    λ m:nat. λ X:Type. False)

Modern Coq

```
Definition EQZ (n:nat) : Type :=
  match n with
  | 0 => True
  | S _ => False
end.
```

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

- pattern-matching
- structural recursion

Universes

In Coq: $\text{Type}_1 : \text{Type}_2 : \text{Type}_3 : \ldots$
together with pattern-matching (R operators) towards all Type_i

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

Martin-Löf's proposal:
An addition type $\mathrm{U}: \mathrm{Type}$ (universe)
"constructors" for this type:

$\mathrm{n}:\mathrm{U}$

$\mathrm{bot} : \mathrm{U}$

$\pi : \Pi\, \mathrm{a}:\mathrm{U}.\ (\mathrm{tr\ a}) \rightarrow \mathrm{U}) \rightarrow \mathrm{U}$

$\ldots$

with:

$\mathrm{tr} : \mathrm{U} \rightarrow \mathrm{Type}$

$\mathrm{tr\ n} \triangleright \mathrm{nat}$

$\mathrm{tr\ bot} \triangleright \bot$

$\mathrm{tr}\ (\pi\ \mathrm{a\ f}) \rightarrow \Pi\, \mathrm{x}:\mathrm{tr\ a}.\ \mathrm{tr(f\ x)}$

Type Universes in Martin-Löf style

Martin-Löf's proposal:

An addition type U: Type (universe)

"constructors" for this type:

$\mathrm{n} : \mathrm{U}$

$\mathrm{bot} : \mathrm{U}$

$\pi : \Pi\ \mathrm{a}{:}\mathrm{U}.\ (\mathrm{tr\ a}) \to \mathrm{U}) \to \mathrm{U}$

$\ldots$

with:

$\mathrm{tr} : \mathrm{U} \to \mathrm{Type}$

$\mathrm{tr\ n} \triangleright \mathrm{nat}$

$\mathrm{tr\ bot} \triangleright \bot$

$\mathrm{tr}\ (\pi\ \mathrm{a\ f}) \to \Pi\ \mathrm{x}{:}\mathrm{tr\ a}.\ \mathrm{tr}(\mathrm{f\ x})$

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 ?

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
```