Universes à la Martin-Löf

Predicative quantification over types in MLTT
Keep MLTT as presented in the course and add:

$U : \mathrm{Type}$

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

$\pi : \Pi A : U, ((\mathrm{tr}\ A) \to U) \to U$

$\mathrm{nat} : U$

$\mathrm{eq} : \Pi A : U, (\mathrm{tr}\ A) \to (\mathrm{tr}\ A) \to U$

$\sigma : \Pi A : U, ((\mathrm{tr}\ A) \to U) \to U$

$\mathrm{sum} : U \to U \to U$

$\mathrm{False} : U$

$\mathrm{tr}\ (\pi\ A\ B) \triangleright \Pi x{:}\mathrm{tr}\ A.\ \mathrm{tr}\ (B\ x)$

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

$\mathrm{tr}\ (\mathrm{eq}\ A\ a\ b) \triangleright a =_A b$

$\mathrm{tr}\ (\sigma\ A\ B) \triangleright \Sigma x{:}\mathrm{tr}\ A.\ \mathrm{tr}\ (B\ x)$

$\mathrm{tr}\ (\mathrm{sum}\ A\ B) \triangleright A+B$

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

$U : \text{Type}$

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

$\pi : \Pi A : U, ((tr\ A) \to U) \to U$

$nat : U$

$eq : \Pi A : U, (tr\ A) \to (tr\ A) \to U$

$\sigma : \Pi A : U, ((tr\ A) \to U) \to U$

$sum : U \to U \to U$

$False : U$

$tr\ (\pi\ A\ B) \triangleright \Pi\ x{:}tr\ A.\ tr\ (B\ x)$

$tr\ nat \triangleright N$

$tr\ (eq\ A\ a\ b) \triangleright a =_A b$

$tr\ (\sigma\ A\ B) \triangleright \Sigma\ x{:}tr\ A.\ tr\ (B\ x)$

$tr\ (sum\ A\ B) \triangleright A+B$

$tr\ False \triangleright \bot$

Idea: if we quantify over U, we quantify over all types ! (except U)

$u : U \quad tr\ u \triangleright U$ would give Type : Type and a paradox

Embedded Universes

U_1 : Type

tr : $U_1 \to$ Type

$\pi : \Pi\, A : U_1, ((\mathrm{tr}\ A) \to U_1) \to U_1$

nat : U_1

eq : $\Pi\, A : U_1, (\mathrm{tr}\ A) \to (\mathrm{tr}\ A) \to U_1$

$\sigma : \Pi\, A : U_1, ((\mathrm{tr}\ A) \to U_1) \to U_1$

sum : $U_1 \to U_1 \to U_1$

False : U_1

u : U_1

tr (π A B) $\rhd$ Π x:tr A. tr (B x)

tr nat $\rhd$ N

tr (eq A a b) $\rhd$ $a =_A b$

tr (σ A B) $\rhd$ Σ x:tr A. tr (B x)

tr (sU1m A B) $\rhd$ A+B

tr False $\rhd$ $\bot$

tr u $\rhd$ U

U comprises all types including U but not U1

Inductive-recursive definition

What is this object U ?

U1 : Type

tr : U1 → Type

Here !

π : Π A : U1, ((tr A)→ U1) → U1   tr (π A B)   ▷   Π x:tr A. tr (B x)

An inductive definition:

- inductive type U
- constructor π
- recursive function tr

It can be viewed as an instance of a powerful extension of the inductive definition scheme

But… the function is used in the type of the constructor !

Using universes

Proving $0 \neq 1$

Not possible in MLTT as given in the course notes
$0=1 \to \bot$ mapped to system T would give a term of type $N \to \bot$

We need a property $P : N \to \text{Type}$ such that $P\ 0 \triangleright T$ and $P\ (S\ x) \triangleright \bot$

How to proceed ?

$Q : N \to U$ $\quad Q\ 0 \triangleright \text{nat}$ and $Q\ (S\ x) \triangleright \text{False}$

then take $P = \lambda x{:}N.\ \text{tr}\ (Q\ x)$

$Q = R_U\ \text{nat}\ \ \lambda p{:}N.\ \lambda R{:}U\ .\ \text{False}$

Universes in Coq are a little different

Digression: computational proofs

The conversion rule

$t : P$

t is of type P
t is a proof of P

$$\frac{t : A \qquad B : Prop}{t : B} \quad A =_c B$$

From the logical point of view, A and B are the *same* proposition

$=_c$ encaptures the computations of the system
for instance, $2+2 =_c 4$

Proofs by computation

We are used to use this rule:

```
0 = 0 + 0        0 = 0
```

```
forall n, n = n + 0
```

```
n = n + 0 -> S n = (S n) + 0

S n = S (n + 0)

S n = S n
```

Combination of computation and deduction

Simple purely computational proof

```
2 + 2  ⇒  4

2 + 2 = 4   ⇒   4 = 4

refl 4 : 4 = 4          refl 4 : 2+2 = 4

                        refl 400 : 200+200 = 4
```

Why is a number prime ?

5 is prime because :

- 2 does not divide 5
- 3 does not divide 5
- 4 does not divide 5
- 0 does not divide 5
- all other natural numbers are either 1, 5, or strictly larger than 5
- and if they are > 5, they do not divide 5

How do we formalize this in Coq ?

A more computational proof

- Write `test : nat -> bool`
- `test n` tries to divide n by 2, 3, … , n-1 and returns true iff it finds no diviso
- prove:

```
test_corr : forall n, test n = true -> prime n
```

what is a proof of `prime 5` ?

```
test_corr 5 (refl true) : prime 5
```

needs to check `refl true : test 5 = true`

needs to compute `test 5` ▸ `true`

Going further

2855425422282796139015635661021640083261642386447028891992474566022844
0039060065387595457150553984323975451391589615029787839937705607143516
9747221107988791198200988477531339214282772016059009904586686254989084
8157354224804090223442975883525260043838906326161240763173874168811485
9248618836187390417578314569601691957439076559828018859903557844859107
7683677175520434074287726578006266759615970759521327828555662781678385
6915818444364448125115624281367424904593632128101802760960881114010033
7757036354572512092407364692157679714619938761929656030268026179011813
2925012323046444438622308877924609373773012481681672424493674474488537
7701557830068808526481615130671448147902883666640622572746652757871273
7464923109637500117090189078626332461957879573142569380507305611967758
0338084333381987500902968831935913095269821311141322393356490178488728
9822881562826008138312961436638459454311440437538215428712777456064478
5856415921332844358020642271469491309176271644704168967807009677359042
9808909616750452927258000843500344831628297089902728649981994387647234
5742762637296948483047509171741861811306885187927486226122933413689280
5663438446664632657247616727566083910565052897571389932021112149579531
1427946254553305387067821067601768750977866100460014602138408448021225
053689054793742003095722096732954750721718115531871310231057902608580607

is prime !

When the computer helps us

Largest known prime number in 1951 : $(2^{148} + 1) / 17$ (44 digits)

today : $2^{82,589,933} - 1$ (24,862,048 digits)

Why such progress ? obvious
But also new mathematics

Pocklington's theorem (1914)

Let $n > 1$ and natural numbers a, $(p_1, \alpha_1), \ldots, (p_k, \alpha_k)$; n is prime if :

$$
\begin{array}{rcll}
p_1 \ldots p_k & \text{are} & \text{prime numbers} & (0) \\
(p_1^{\alpha_1} \ldots p_k^{\alpha_k}) & | & (n-1) & (1) \\
a^{n-1} & = & 1 (\text{mod } n) & (2) \\
\forall i \in \{1, \ldots, k\}\ \text{gcd}(a^{\frac{n-1}{p_i}} - 1, n) & = & 1 & (3) \\
p_1^{\alpha_1} \ldots p_k^{\alpha_k} & > & \sqrt{n}. & (4)
\end{array}
$$

$a, p_1, \alpha_1 \ldots, p_k, \alpha_k$ is a Pocklington *certificate* for n.

Plan of action

- prove Pocklington's theorem : done by Oostdijk and Caprotti (2001)
- define a data-structure for representing certificates
- write a certificate checker in Coq, prove it correct
- build certificates outside Coq
- Sit back and relax

Defining certificates

A certificate for n is some tupple : $a, p_1, \alpha_1, \ldots, p_k, \alpha_k$.

self-contained certificate : recursively add certificates for each p_i :

$$c = \{n, a, [c_1^{\alpha_1}; \ldots; c_k^{\alpha_k}]\}$$

a certificate for 127 is :

$$\{127, 3, [\{7, 2, [\{3, 2, [(2, \text{prime2})]\}; (2, \text{prime2})]\};$$
$$\{3, 2, [(2, \text{prime2})]\};$$
$$(2, \text{prime2})]\}$$

Formalizing certificates

Share the certificates by flattening the list :

$$[\{127, 3, [7; 3; 2]\}; \{7, 2, [3; 2]\}; \{3, 2, [2]\}; (2, \mathsf{prime2})].$$

such a certificate is a mini-data-base containing all prime numbers used in proving that n is prime.

Checking certificates

$$\forall l,\ \text{Check } l = true \Rightarrow \forall c \in l,\ \text{prime } (n\ c)$$

recursion over the list (certificate) ; test the computational conditions.

only difficulty : time&space of the calculations

```
Inductive positive : Set :=
 | xH : positive
 | xO : positive -> positive
 | xI : positive -> positive.
```

$a^{n-1} = 1 (\text{mod } n)$ main trick : keep things small by calculating modulo n

How are certificates built ?

a C program builds the certificate and prints it as a Coq term.

Different recipes :

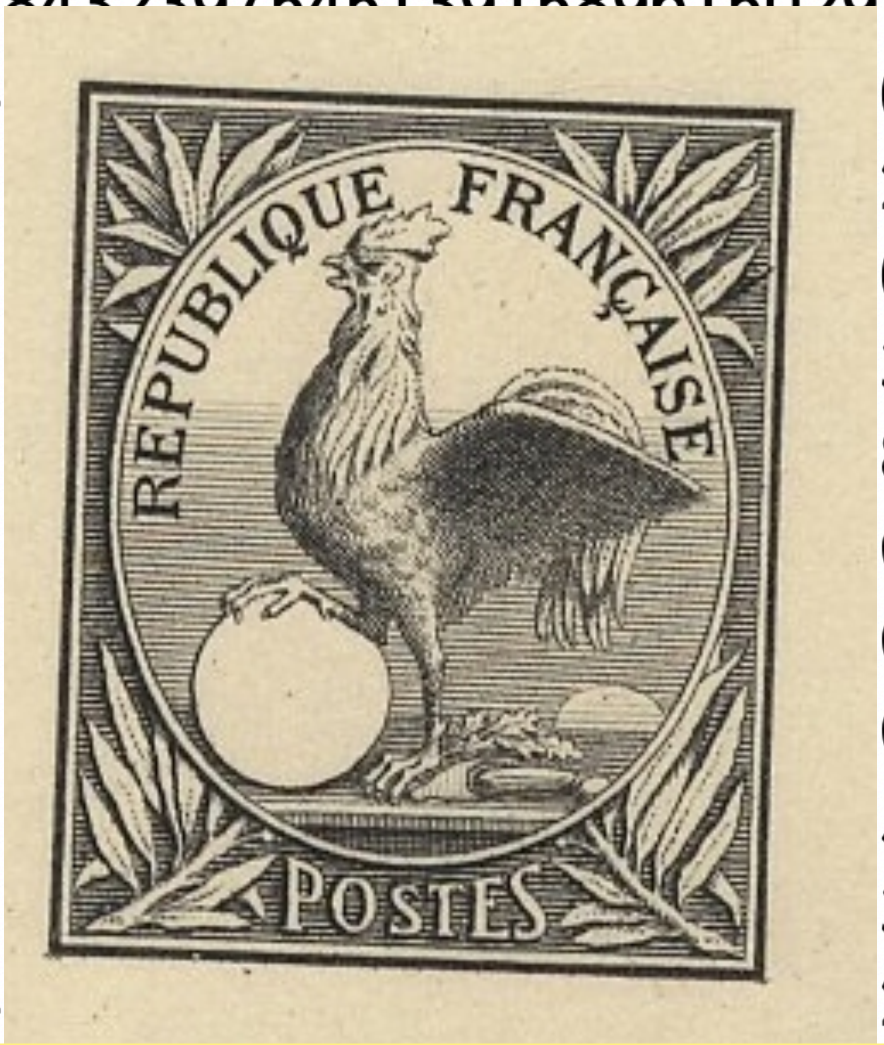
- generic : find a factorization using ECM (Elliptic Curve Library)
- Mersenne : for $2^m - 1$. various tricks $2^n - 1 - 1 = 2(2^{n-1} - 1)$ and $2^{2p} - 1 = (2^p - 1)(2^p + 1)$, $2^{3p} - 1 = (2^p - 1)(2^{2p} + 2^p + 1)$; help find a decomposition.
- Lucas criterion
- Proth numbers

Can be the critical step.

For random prime numbers, up to 200 digits.

For the largest Mersenne primes we treat, some hack was needed.

2855425422282796139015635661021640083261642386447028891992474566022844
0039060065387595457150553984323975451391589615029787839937705607143516
97472211079887911982009884 009904586686254989084
81573542248040902234429758 240763173874168811485
92486188361873904175783145 018859903557844859107
76836771755204340742877265 327828555662781678385
69158184443644481251156242 80276096088114010033
77570363545725120924073646 656030268026179011813
29250123230464444386223088 67242449367447488537
77015578300688085264816151 622572746652757871273
74649231096375001170901890 569380507305611967758
03380843333819875009029688 32239335649017848728
98228815628260081383129614 215428712777456064478
585641592133284435802064 3967807009677359042
980890961675045292725800 3649981994387647234
5742762637296948483047509171741861811306885187927486226122933413689280
5663438446664632657247616727566083910565052897571389932021112149579531
1427946254553305387067821067601768750977866100460014602138408448021225
053689054793742003095722096732954750721718115531871310231057902608580607



proved in Coq!

is prime !

Going further

This is actually old. Since more technology has been brought in:

- more efficient coding of numbers in Coq
- add more efficient representation of these numbers to Coq
- using more modern results about prime numbers (elliptic curves)

It is not just about the numbers

Some theorems seem non-computational in nature ; yet their (known) proofs rely on heavy computations.

- The four color theorem (1976)   done in Coq
- The Kepler conjecture (Thomas Hales, 1998)

Interesting because the exposition of the arguments mixes mathematics and ad-hoc programs ; both sophisticated.

there is a real problem of verification standarts