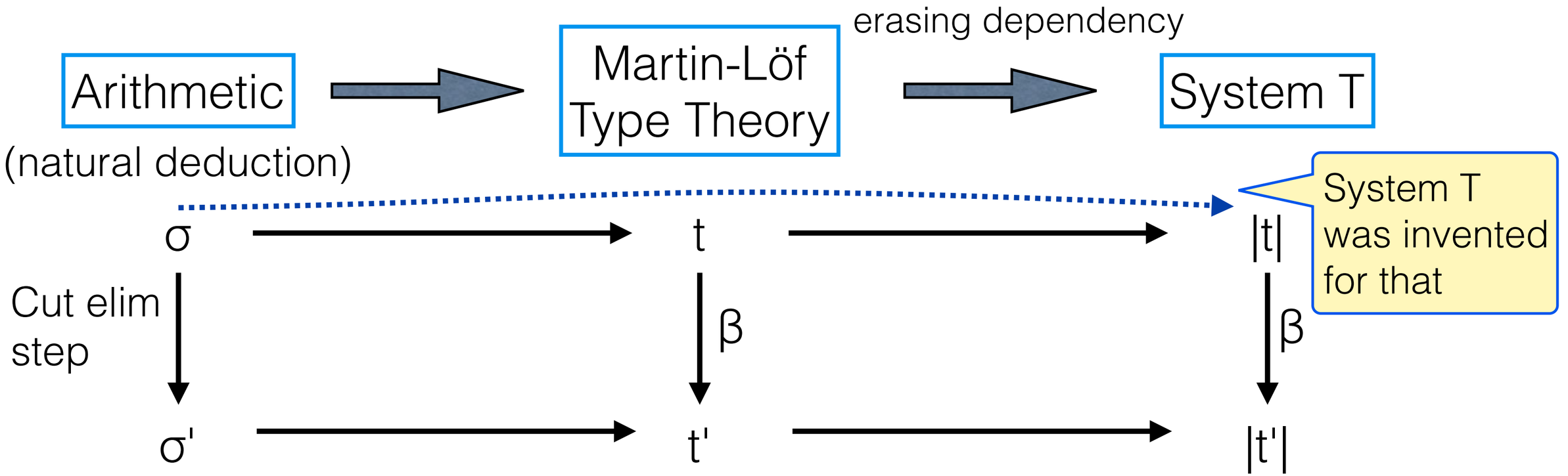
MPRI
PRFSYS

Nov. 4th

System F, PTSs
-
Computational Proofs

How we proved cut elimination



What about cut elimination for HOL ? (Higher-Order Arithmetic, 2nd order arithmetic…)

System F (à la Church)

$$T ::= \alpha \mid T \to T \mid \forall \alpha . T$$

$$t ::= x \mid t\ t \mid \lambda x : T . t \mid \Lambda \alpha . T \mid t\ T$$

$$\frac{}{\Gamma \vdash x : T}\ (x:T) \in \Gamma \qquad \frac{\Gamma \vdash u : T \to U \qquad \Gamma \vdash t : T}{\Gamma \vdash u\ t : U} \qquad \frac{\Gamma\ (x:T) \vdash u : U}{\Gamma \vdash \lambda x:T . u : T \to U}$$

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash \Lambda \alpha . t : \forall \alpha . T}\ \alpha \notin FV(\Gamma) \qquad \frac{\Gamma \vdash t : \forall \alpha . T}{\Gamma \vdash t\ U : T[U \setminus \alpha]}$$

Type inference is decidable: given Γ and t, there exists T s.t. $\Gamma \vdash t : T$

System F (à la Curry)

$$T ::= \alpha \mid T \to T \mid \forall \alpha . T$$

$$t ::= x \mid t\ t \mid \lambda x . t$$

$$\frac{}{\Gamma \vdash x : T}\ (x:T) \in \Gamma$$

$$\frac{\Gamma \vdash u : T \to U \qquad \Gamma \vdash t : T}{\Gamma \vdash u\ t : U}$$

$$\frac{\Gamma\ (x:T) \vdash u : U}{\Gamma \vdash \lambda x . u : T \to U}$$

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash t : \forall \alpha . T}\ \alpha \notin FV(\Gamma)$$

$$\frac{\Gamma \vdash t : \forall \alpha . T}{\Gamma \vdash t : T[U \setminus \alpha]}$$

Type inference is undecidable for this version

Impredicativity for Datatypes

$A \times B \equiv \forall X . (A \to B \to X) \to X$

$(a,b) \equiv \Lambda X. \lambda f : A \to B \to X. f\ a\ b$

$\pi_1 \equiv \lambda c : A \times B.\ \ c\ A\ (\lambda a{:}A . \lambda b : B . a)$

$\pi_2 \equiv \lambda c : A \times B.\ \ c\ B\ (\lambda a{:}A . \lambda b : B . b)$

$unit \equiv \forall X. X \to X$

$() \equiv \Lambda X . \lambda x{:}X . X$

$\bot \equiv \forall X . X$

$A + B \equiv \forall X . (A \to X) \to (B \to X) \to X$

$i(a) \equiv \Lambda X. \lambda f : A \to X. \lambda g : B \to X . f\ a$

$j(b) \equiv \Lambda X. \lambda f : A \to X. \lambda g : B \to X . g\ b$

$\delta(c, f, g) \equiv c\ C\ f\ g$

Impredicativity for Church numerals

Church ≡ ∀X . X → (X→X) → X

0 ≡ Λ X. λ x:X. λ f: X→X. x

1 ≡ Λ X. λ x:X. λ f: X→X. f x

2 ≡ Λ X. λ x:X. λ f: X→X. f (f x)

…

S ≡ λ n:Church . Λ X. λ x:X. λ f: X→X. f (n Church x f)

add≡ λ n:Church . λ m:Church . (n Church (m Church) S)

R ≡ ?

pred ≡ ?

Impredicativity for other recursive datatypes

$\text{List}_A \equiv \forall X\,.\, X \to (A \to X \to X) \to X$

$\text{nil} \equiv \Lambda\, X.\ \lambda\, x{:}X.\ \lambda\, f{:}\, A \to X \to X.\ x$

…

$\text{cons} \equiv \lambda\, a{:}A\,.\,\lambda\, l{:}\text{List}_A\,.\,\Lambda\, X.\ \lambda\, x{:}X.\ \lambda\, f{:}\, A \to X \to X.\ f\ a\ (l\ \text{List}_A\ x\ f)$

$\text{bintree} \equiv \forall X\,.\, X \to (A \to X \to X \to X) \to X$

$\text{Ord} \equiv \forall X\,.\, X \to (X \to X) \to ((\text{Church} \to X) \to X) \to X$

From F to Fω

F for cut elimination of 2nd order arithmetic (can quantify over propositions, predicates, but not properties of properties)

A:Type, P : nat→Type are mapped to Types (after erasing dependency)

CR : (term→Type)→Type would be mapped to some Type→Type

In Fω : simple calculus of types

$$\mathrm{K} ::= \mathrm{Type} \mid \mathrm{K}\to\mathrm{K}$$

$$\mathrm{T} ::= \alpha \mid \mathrm{T}\to \mathrm{T} \mid \forall\, \alpha{:}\mathrm{K}.\mathrm{T} \mid \Lambda\, \alpha{:}\mathrm{K}.\mathrm{T} \mid \mathrm{T}\ \mathrm{T}$$

$$\mathrm{t} ::= \mathrm{x} \mid \lambda\, \mathrm{x}{:}\mathrm{T}.\mathrm{t} \mid \mathrm{t}\ \mathrm{t} \mid \Lambda\, \alpha{:}\mathrm{K}.\mathrm{t} \mid \mathrm{t}\ \mathrm{T}$$

Fω rules

$$\frac{\Gamma \text{ wf}}{\Gamma\ (\alpha{:}K)\ \text{wf}} \qquad \frac{\Gamma \text{ wf}}{\Gamma \vdash \alpha{:}K} \qquad \frac{\Gamma \vdash T_1{:}\text{Type} \quad \Gamma \vdash T_1{:}\text{Type}}{\Gamma \vdash T_1 \to T_2{:}\text{Type}} \qquad \frac{\Gamma\ (\alpha{:}K) \vdash T{:}\text{Type}}{\Gamma \vdash \forall\ \alpha{:}K.T{:}\text{Type}}$$

$$\frac{\Gamma\ (\alpha{:}K_1) \vdash T{:}K_2}{\Gamma \vdash \Lambda\ \alpha{:}K_1.T : K_1 \to K_2} \qquad \frac{\Gamma \vdash T_1{:}K_1 \qquad \Gamma \vdash T_2 : K_1 \to K_2}{\Gamma \vdash T_1\ T_2 : K_2}$$

$$\frac{\Gamma \vdash T : \text{Type}}{\Gamma\ (x{:}T)\ \text{wf}} \qquad \frac{\Gamma \text{ wf}}{\Gamma \vdash x : T}\ (x{:}T) \qquad \frac{\Gamma\ (x{:}U) \vdash t : T}{\Gamma \vdash \lambda\ x{:}U.t : U \to T}$$

$$\frac{\Gamma \vdash t : U \to T \quad \Gamma \vdash u : U}{\Gamma \vdash t\ u{:}\ T} \qquad \frac{\Gamma(\alpha{:}K) \vdash t : T}{\Gamma \vdash \Lambda\ \alpha{:}K.t{:}\forall\alpha{:}K.T} \qquad \frac{\Gamma \vdash t{:}\forall\alpha{:}K.T \quad \Gamma \vdash U{:}K}{\Gamma \vdash t\ U : T[\alpha \setminus U]}$$

$$\Lambda\ \alpha{:}K.T\ U \rhd_\beta T\ [\alpha \setminus U]$$

From a programming language point of view

We can talk about parametrized types:

$$\mathrm{list} : \mathrm{Type} \to \mathrm{Type}$$

$$\mathrm{list} \equiv \quad \Lambda\alpha{:}\mathrm{Type}.\ \forall\ \mathrm{X}{:}\mathrm{Type}.\ \mathrm{X} \to(\alpha \to\mathrm{X} \to\mathrm{X}) \to\mathrm{X}$$

From a programming language point of view:

- the impredicative encoding is not practically useful
- but talking about functions from types to types is

What happens when we add F / Fω
to our type dependent systems ?

PTSs

Sorts $\mathcal{S}$

$\mathcal{S} = \{\text{Type; Kind}\}$ (a.o.)

$\mathcal{A} \subset \mathcal{S}\text{x}\mathcal{S}$ $(= \{(\text{Type,Kind})\})$

$\mathcal{R} \subset \mathcal{S}\text{x}\mathcal{S}$

$$t ::= s \mid x \mid \lambda\, x{:}t.t \mid t\ t \mid \Pi\, x{:}T.t$$

$$[]\ \text{wf} \qquad \frac{\Gamma \vdash T{:}s}{\Gamma(x{:}T)\ \text{wf}} \qquad \frac{\Gamma\ \text{wf}}{\Gamma \vdash x{:}T}$$

$$\frac{\Gamma\ \text{wf}}{\Gamma \vdash s_1{:}s_2}\,(s_1,s_2) \in \mathcal{A}$$

$$\frac{\Gamma \vdash T : s_1 \quad \Gamma(x{:}T) \vdash U{:}s_2}{\Gamma \vdash \Pi\, x{:}T.U : s_2}\,(s_1,s_2) \in \mathcal{R}$$

$$\frac{\Gamma \vdash \Pi\, x{:}T.U : s \quad \Gamma(x{:}T) \vdash t : U}{\Gamma \vdash \lambda x{:}T.t{:}\Pi\, x{:}T.U}$$

$$\frac{\Gamma \vdash u : \Pi\, x{:}T.U \quad \Gamma \vdash t : T}{\Gamma \vdash u\ t : U[x \setminus t]}$$

$$\frac{\Gamma \vdash t{:}T \quad \Gamma \vdash T'{:}s}{\Gamma \vdash t : T'}\ T =_\beta T'$$

PTSs

Sorts S

$t ::= s \mid x \mid \lambda\, x{:}t.t \mid t\ t \mid \Pi\, x{:}T.t$

Γ wf

$\Gamma \vdash s_1 : s_2$

$$\frac{\Gamma \vdash T : s_1 \quad \Gamma(x:T) \vdash U : s_2}{\Gamma \vdash \Pi\, x{:}T.U : s_2}$$

R = {(Type, Type)}
{(Type, Type); (Type, Kind)}
{(Type, Type); (Kind, Type)}
{(Type, Type); (Kind, Type); (Kind,Kind)}
{(Type, Type); (Kind, Type); (Kind,Kind); (Type, Kind)}

Simply typed calculus $\lambda\!\rightarrow$

LF aka $\lambda\,\Pi$

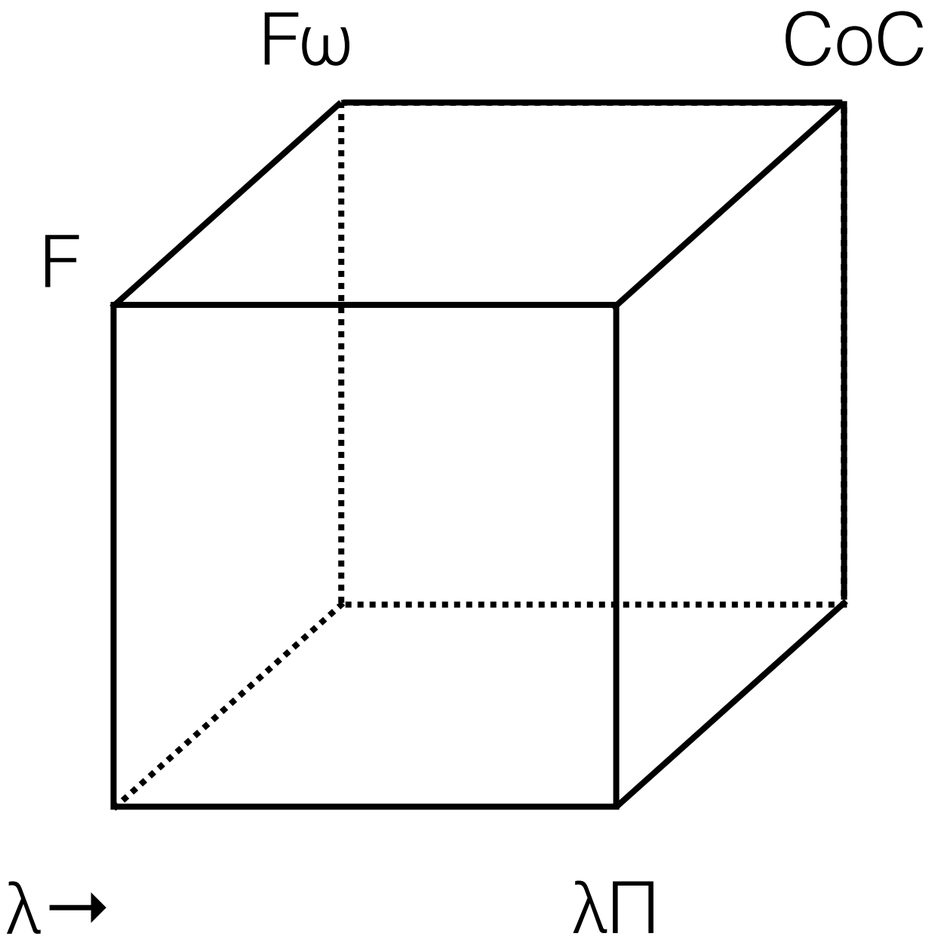
System F

Fω

Calculus of Constructions

Calculus of Constructions: Coquand & Huet, 1985

Barendregt's Cube



Dimensions:

- Impredicativity
- Type constructors (?)
- Dependent Types

Other PTSs

$$\frac{\Gamma \vdash T : s_1 \qquad \Gamma(x{:}T) \vdash U : s_2}{\Gamma \vdash \Pi\, x{:}T.U : s_3}\ (s_1, s_2, s_3) \in \mathcal{R}$$

Original terminology:
Extended Calculus of Constructions
(ECC, Luo)

Underlying Rocq:

$$\mathrm{Prop} : \mathrm{Type}_1 : \mathrm{Type}_2 : \mathrm{Type}_3 : \ldots$$

Rules: $(\mathrm{Type}_i, \mathrm{Prop}, \mathrm{Prop})$ (polymorphism / impredicativity)
$(\mathrm{Type}_i, \mathrm{Type}_j, \mathrm{Type}_{\max(i,j)})$ (predicative universes)

Additional subtyping: $\mathrm{Type}_i \subset \mathrm{Type}_{i+1}$
covariance: $T \subset U \Rightarrow \Pi\, x{:}A.T \subset \Pi\, x : A.U$

Other additions: inductive, co-inductive types…

Simple Model of Rocq

Without going into details
We mimick everything in Set Theory:

$$|\mathrm{Prop}| = \{0; 1\}$$
$$|\Pi\ \mathrm{x{:}A.B}|_I = \prod_{\alpha \in |A|} |B|_{I;\ x \leftarrow \alpha}$$
$$|\lambda\ \mathrm{x{:}T.t}|_I = \quad \alpha \in |T|_I \quad \mapsto \quad |t|_{I;\ x \rightarrow \alpha}$$

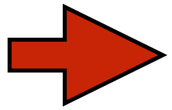
etc…

- Not deep
- Some technical details must be taken care of
- Allows to easily validate some axioms
- Not possible with an impredicative sort Set in which 0≠1 !

Computational Proofs

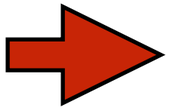
Simple purely computational proof

```
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 Rocq ?

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

2855425422827961390156356610216400832616423864470288919924745660228440039060065387595457150553984323975451391589615029787839937056071435169747221107988791198200988477531339214282772016059009904586686254989084815735422480409022344297588352526004383890632616124076317387416881148592486188361873904175783145696016919574390765598280188599035578448591077683677175520434074287726578006266759615970759521327828555662781678385691581844436444812511562428136742490459363212810180276096088111401003377570363545725120924073646921576797146199387619296560302680261790118132925012323046444438622308877924609373773012481681672424493674474488537770155783006880852648161513067144814790288366664062257274665275787127374649231096375001170901890786263324619578795731425693805073056119677580338084333381987500902968831935913095269821311141322393356490178488728982288156282600813831296143663845945431144043753821542871277745606447858641592133284435802064227146949130917627164470416896780700967735904298089096167504529272580008435003448316282970899027286499819943876472345742762637296948483047509171741861811306885187927486226122933413689280566343844666463265724761672756608391056505289757138993202111214957953114279462545533053870678210676017687509778661004600146021384084480212250536890547937420030957220967329547507217181155318713102310579026085806070

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 :

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

$a, p_1, \alpha_1 \ldots, p_k, \alpha_k$ 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 :

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

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.

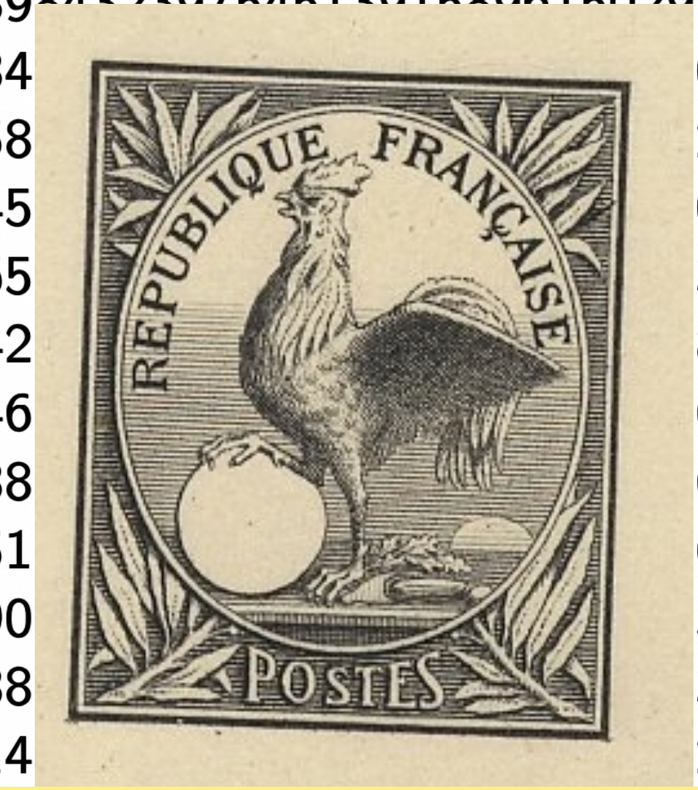
Different recipes :

- 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.

28554254222827961390156356610216400832616423864470288919924745660228440039060065387595457150553984323075451301580615029787839937705607143516974722110798879119820098840099045866862549890848157354224804090223442975824076317387416881148592486188361873904175783145018859903557844859107768367717552043407428772653278285556627816783856915818444364448125115624280276096088111401003377570363545725120924073646656030268026179011813292501232304644443862230886724244936744744885377701557830068808526481615162257274665275787127356469238096375001170901890569380507305611967758033808433338198750090296883223933564901784887289822881562826008138312961421542871277745606447858564159213328443580206467807009677359042980890961675045292725800499819943876472345742762637296948483047509171741861811306885187927486226122933413689280566343844666463265724761672756608391056505289757138993202111214957953114279462545533053870678210676017687509778661004600146021384084480212250536890547937420030957220967329547507217181155318713102310579026058060 

proved in Rocq!

is prime !

Going further

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

- more efficient coding of numbers in Rocq
- add more efficient representation of these numbers to Rocq
- 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

Computational Reflection

Using computation can also be useful for some automations. Like the Ring tactic