MPRI
PRFSYS

Nov. 18th

Computational and less computational sorts

$\not\in$

The sorts of Rocq

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

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…

Why the additions ?

Inductive types in order to have:

- more efficient representations / computations
- dependent elimination schemes (induction…)
- 0 ≠ 1 thanks to "large elimination"

```
Definition EQZ n : Prop :=
   match n with
    | O    => True
    | S _  => False
 end.
```

What about Set ?

Originally : CoC, that is Prop and Kind

Prop then duplicated into Prop and Set
original motivation: Program extraction : programming by proving

permutation

$$\mathrm{p} : \ \Pi\ \mathrm{l} : \mathrm{list}\ .\ \Sigma\ \mathrm{l'} : \mathrm{list}\ .\ \ \mathrm{sorted\ l'}\ \ \wedge\ \ \ \mathrm{l} \simeq \mathrm{l'}$$

$$\mathrm{sort} \equiv \lambda\ \mathrm{l}\ .\ \pi_1(\mathrm{p\ l}) : \quad \mathrm{list} \rightarrow \mathrm{list}$$

$$\lambda\ \mathrm{l}\ .\ \pi_2(\mathrm{p\ l}) : \quad \Pi\ \mathrm{l} : \mathrm{list}\ .\ \mathrm{sorted}\ (\mathrm{sort\ l})\ \ \wedge\ \ \ \mathrm{l} \simeq (\mathrm{sort\ l})$$

But we want to erase as much of p as we can: everything that is not used to compute l'

using sorts to mark terms

$$\frac{\Gamma \vdash T : s}{\Gamma\ (x{:}T)\ \mathrm{wf}}$$

if s is Set: computational (kept)
if s is Prop: only specification (erased when extracting)

Reflected in the elimination rules for inductives:

```
Definition EQZ n : Prop :=
   match n with
    | O    => True
    | S _  => False
 end.
```

authorized because **`nat:Set`**

```
Inductive nat : Prop :=
| O
| S : nat -> nat.
```

we cannot prove O ≠ 1 for these "nats"

These limitations ensure that the extracted term is well-typed (has no "holes")

A very short story of extraction

Originally: novel way to write certified programs. In line with Curry-Howard

$$\mathrm{p} : \ \Pi\ \mathrm{l} : \mathrm{list}\ .\ \Sigma\ \mathrm{l'} : \mathrm{list}\ .\ \ \mathrm{sorted}\ \mathrm{l'}\ \ \wedge\ \ \ \mathrm{l} \simeq \mathrm{l'}$$

$$\mathcal{E}(\mathrm{p}) : \mathrm{list} \to \mathrm{list}$$

$$\mathcal{R}(\mathrm{p}) : \ \Pi\ \mathrm{l} : \mathrm{list}\ .\ \mathrm{sorted}\ (\mathcal{E}(\mathrm{p})\ \mathrm{l})\ \ \wedge\ \ \ \mathrm{l} \simeq (\mathcal{E}(\mathrm{p})\ \mathrm{l})$$

Then (PhD of JC Filiâtre):

From $\mathrm{sort} : \mathrm{list} \to \mathrm{list}$ and specification $\Pi\ \mathrm{l} : \mathrm{list}\ .\ \Sigma\ \mathrm{l'} : \mathrm{list}\ .\ \ \mathrm{sorted}\ \mathrm{l'}\ \ \wedge\ \ \ \mathrm{l} \simeq \mathrm{l'}$

generate the assumptions to retrieve p

Base of program tactic, then of Why (Why3) etc…

Next step: allowing some classical axioms

```
A ∨ B : Prop
A + B : Set
```

```
forall A : Set, A + (A→⊥)
```
problematic

```
forall A : Set, A \/ (A→⊥)
```
can be considered ok

Such considerations became a main motivation for Set / Prop distinction

```
forall A : Set, A + (A→⊥)
```
actually much more problematic

not only for computations, but also for consistency

A limitation of (some) type theories

$$J : \prod A : \mathrm{Type} . \prod P : A \to \mathrm{Type} . \prod x\ y : A . x=y \to P\,x \to P\,y$$

$$J\ A\ P\ t\ u\ (\mathrm{refl}_A\ \_)\ p \ \triangleright\ p$$

Thomas Streicher identified the need for a second axiom/operator:

$$K : \prod A : \mathrm{Type} . \prod x : A . \prod P : x=x \to \mathrm{Type} . P(\mathrm{refl}_A\ x) \to \prod h : x=x . P\,h$$

Basically: there is only one (canonical) equality proof

$$K\ A\ t\ (\mathrm{refl}_A\ x)\ p \triangleright p$$

- Cannot be proved in Rocq
- Some versions of Agda allow to prove K

Why is it important ?

With J we routinely construct:

$$J\ A\ P\ t\ u\ :\ t{=}u \to P\,t \to P\,u$$

Think of P as an actual (dependent) type: this is a translation from P t to P u

so we want: J A P t t e p = p

We have J A P t t (refl_A t) p $\triangleright$ p.

To prove this we need e= (refl_A t)

- The reduction for K is not too important. Assuming K as an axiom is generally enough
- Sometimes one does not want K (HoTT…)

Why do I mention K here ?

1. It is another useful extension of Type Theory
2. It is linked to the excluded middle
3. We will use it for an example / exercise

2: the excluded middle allows to prove K

Actually, if $\Pi\ x\ y : A, x{=}y \vee x{\neq}\ y,$ then one can prove uniqueness of equality proofs for A

Not difficult but a little technical (you can see in the exercise)

Excluded middle and impredicativity

If Prop is impredicative

and we have decidable equality (a fortiori excluded-middle) then we can prove proof-irrelevance:

$$\Pi\ \mathrm{P{:}Prop}.\ \Pi\ \mathrm{p_1}\ \mathrm{p_2} : \mathrm{P}.\ \mathrm{p_1} = \mathrm{p_2}$$

In other words:

- we can assume EM in Prop, provided we have no large elimination (that is we cannot prove 0≠1, true≠false… for nat or bool : Prop)
- we cannot assume EM in an impredicative sort Set

A variant of the Barbanera-Berardi result

We prove proof-irrelevance in the calculus of constructions with:

- decidable equality (a fortiori EM)
- inductive equality with the regular elimination scheme
- equality over Kind (means we need one more universe)

- Makes the point that it is useful to have the possibility to tag computational and non-computational type
- A cute (I think) little variation on Russell's paradox

A paradox

We take:

$\mathrm{Bool} : \mathrm{Prop}$

$\mathrm{t} : \mathrm{Bool} \quad \mathrm{f} : \mathrm{Bool}$

Let us define :

$\mathrm{U} \equiv \Pi\ \mathrm{P} : \mathrm{Prop}.\ \mathrm{P} \to \mathrm{Bool} \ : \mathrm{Prop}$ (we use impredicativity)

Idea: $u_1 \in u_2 \equiv u_1\ U\ u_2 = t$

Now, given $\mathrm{P} : \mathrm{U} \to \mathrm{Bool}$ can we define the U corresponding to P ?

$\{\mathrm{u}{:}\mathrm{U} \mid \mathrm{P}\ \mathrm{u} = \mathrm{t}\}$

Given $\mathrm{P} : \mathrm{U} \to \mathrm{Bool}$ can we define the U corresponding to P ?

$$\{\mathrm{u}{:}\mathrm{U} \mid \mathrm{P}\ \mathrm{u} = \mathrm{t}\}$$

We need to turn $\mathrm{U} \to \mathrm{Bool}$ into $\Pi\ \mathrm{P} : \mathrm{Prop}.\ \mathrm{P} \to \mathrm{Bool}$

that is, given $\mathrm{P} : \mathrm{Prop}$ into $\mathrm{P} \to \mathrm{Bool}$

We use decidability of equality:

- if P=U, we transform $\mathrm{U} \to \mathrm{Bool}$ into $\mathrm{P} \to \mathrm{Bool}$
- if not, anything goes (always return f for instance)

Gives us a form of comprehension:
$\mathrm{Comp} : (\mathrm{U} \to \mathrm{Bool}) \to \mathrm{U}$

We can show: $(\mathrm{Comp}\ \mathrm{P}\ \mathrm{U}\ \mathrm{u}) = \mathrm{P}\ \mathrm{u}$

(here we need to use uniqueness of equality)

A paradox (3)

$$(\mathrm{Comp}\ \mathrm{P}\ \mathrm{U}\ \mathrm{u}) = \mathrm{P}\ \mathrm{u}$$

We can define "negation" $\mathrm{nb} : \mathrm{Bool} \to \mathrm{Bool}$

$$\mathrm{nb}\ \mathrm{t} = \mathrm{f}$$
$$\mathrm{nb}\ \mathrm{f} = \mathrm{t}$$

How ? using EM / decidability of equality !

We can then define Russell's set: $\mathrm{R} = \{\mathrm{x} : \mathrm{U} \mid \mathrm{x}\ \mathrm{U}\ \mathrm{x} = \mathrm{f}\}$

So… $\mathrm{R}\ \mathrm{U}\ \mathrm{R} = \mathrm{nb}\ (\mathrm{R}\ \mathrm{U}\ \mathrm{R})$

From this, we deduce $\mathrm{t} = \mathrm{f}$