Universe Polymorphism in Coq, for the OCAML hacker

Matthieu Sozeau, Inria Paris & IRIF

Coq Implementors Workshop 2016
June 2nd 2016
Sophia-Antipolis
What are universes?

Universe Polymorphism in Coq, for the OCaml hacker

Matthieu Sozeau - Universe Polymorphism in Coq, for the OCaml hacker
How are they organised?

A hierarchy of predicative universes $\text{Type}_0 < \text{Type}_1 < \ldots$

- Avoids the $\text{Type} : \text{Type}$ paradox (system $U^-$)
- Replicates Russell’s paradox of $\{x \mid x \notin x\}$, the set of all sets etc....
- Think of $\text{Type}_0$ as sets, $\text{Type}_1$ as classes etc....
sort of $t = \text{type of the type of } t$, necessarily a $\text{Type}_i$.

\[
\begin{align*}
\text{TYPE-INTRO} & \quad \text{TYPE-PROD} \\
\Gamma \vdash i \in \mathbb{N} & \\n\Gamma \vdash \text{Type}_i : \text{Type}_{i+1} & \quad \Gamma \vdash A : \text{Type}_i, x : A \vdash B : \text{Type}_j \\n\Gamma \vdash \Pi x : A. B : \text{Type}_{\max(i,j)} & 
\end{align*}
\]

type Level.t \\
type Universe.t = (Level.t * int) list (* max([i(+n?)]) *)
Typical ambiguity

Working with explicit universe indices is cumbersome, annotations pervade definitions and proofs.

⇒ Allow *typical ambiguity* (first used by Russell in Principia).

Idea: write **Type** to mean any type that “fits” (keeps the system consistent).

➤ On paper: let the reader infer levels for universes and check consistency.

➤ On computer: let the computer infer levels and check consistency in the background.
Floating universes

Formally, translate from anonymous *Types* to explicit *Type*\(_i\)s.
But in general many \(i\)'s can work!

\[
\text{Definition } \text{id } (A : \text{Type}) (a : A) := a.
\]

\[
\vdash \text{id} : \Pi(A : \text{Type}_0), A \rightarrow A : \text{Type}_1
\]

or

\[
\vdash \text{id} : \Pi(A : \text{Type}_1), A \rightarrow A : \text{Type}_2
\]

or ...?

\[\Rightarrow \text{universe variables}\]

\[
\text{type Level.t = Prop | Set} \\
| \text{Level of int} \times \text{DirPath.t} (* \text{global} *)
\]
Floating universes and constraints

Consistency is now ensured by giving an assignment of natural numbers to universe variables, satisfying constraints. New judgment $\vdash_{\text{float}}$

\[
\text{TYPE-INTRO} \quad \frac{} {\Gamma \vdash_{\text{float}} \Gamma \ (i, j \in \mathbb{L})} \\
\frac{} {\Gamma \vdash_{\text{float}} \text{Type}_i : \text{Type}_j \rightsquigarrow i < j}
\]

\[
\text{TYPE-PROD} \quad \frac{\Gamma \vdash_{\text{float}} A : \text{Type}_i \quad \Gamma, x : A \vdash B : \text{Type}_j} {\Gamma \vdash_{\text{float}} \prod x : A. B : \text{Type}_k \rightsquigarrow \max(i, j) \leq k}
\]

type constraint_type = Lt | Le | Eq

type univ_constraint = Level.t * constraint_type * Level.t

module Constraint.t : Set.S with type elt = univ_constraint
Type-checking generates constraints between *algebraic* universes (Universe.t). In the kernel (uGraph.ml):

- can *check* any algebraic universe constraint.
- can only *enforce* atomic constraints between levels (Level.t):
  *anomaly* on non-atomic constraints.

Enforcing constraints of the form $l \leq \max(i, j)$ would require a more complex constraint checking algorithm.
Invariant: only generate constraints of the form \( \max(is) \leq l \) where \( l \) is a level. `Univ.enforce_(l)eq` transforms non-atomic to atomic constraints

- Type inference naturally enforces this (subtyping rule on products being equivariant on the domain, covariant on the codomain).

- Algebraic universes can appear only at the conclusion of the term in type position of the typing judgment. So, when putting an inferred type in a term, one has to `refresh` universes (`Evarsolve.refresh_universes`). Sometimes necessary in tactics.
Without polymorphism

Floating levels provide a restricted kind of polymorphism:

\[
\text{Definition } \text{id} \ (A : \text{Type}) \ (a : A) := a
\]
\[
\leadsto \vdash \text{id} : \Pi(A : \text{Type}_l), \ A \to A : \text{Type}_{l+1}
\]

⇒ \(l\) is \textit{not} quantified at the definition level here, it is \textit{global}:

\[
\not\vdash \text{id} \ (\Pi(A : \text{Type}_l), \ A \to A) \ \text{id} : \tau
\]

Because \(l + 1 \not\leq l\). However \(l\) can gradually move up as high as wanted.
Bounded polymorphism:

Polymorphic Definition \( \text{id} \ (A : \text{Type}) \ (a : A) := a \)

\[ \text{id}_l : \Pi(A : \text{Type}_l), \ A \rightarrow A \]

\( \Rightarrow \) \( l \) is quantified at the definition level now and we can instantiate it at each application:

\[ l < k \vdash_{\text{poly}} \text{id}_k (\Pi(A : \text{Type}_l), \ A \rightarrow A) \text{id}_l : (\Pi(A : \text{Type}_l), \ A \rightarrow A) \]
1 Introduction

2 Elaborating Universes
   - Kernel
   - Engine
   - Unification
   - Minimization
Constraint checking

Constraints are generated once at refinement time outside the kernel. The kernel just checks that the constraints are consistent and sufficient to typecheck the terms.

universe context \( \Psi \) :::= \( i \vdash \Theta \)
Univ.UContext.t = Level.t array * constraints
Univ.ContextSet.t = LSet.t * constraints
Constraints are generated once at refinement time outside the kernel. The kernel just checks that the constraints are consistent and sufficient to typecheck the terms.

universe context \( \Psi ::= \langle i \vdash \Theta \rangle \)

Univ.UContext.t = Level.t array * constraints
Univ.ContextSet.t = LSet.t * constraints

Elaboration in bidirectional fashion:

- **Inference:** \( \Gamma; \Psi \vdash t \uparrow \rightsquigarrow \Psi' \vdash t' : T \)
- **Checking:** \( \Gamma; \Psi \vdash t \downarrow T \rightsquigarrow \Psi' \vdash t' : T \)

Pretyping.pretype, Typing.infer, Typing.check
Constraint checking

Constraints are generated once at refinement time outside the kernel. The kernel just checks that the constraints are consistent and sufficient to typecheck the terms.

\[
\text{universe context} \quad \Psi ::= \vec{i} \vdash \Theta \\
\text{Univ.UContext.t} = \text{Level.t array} \ast \text{constraints} \\
\text{Univ.ContextSet.t} = \text{LSet.t} \ast \text{constraints}
\]

Elaboration in bidirectional fashion:

- **Inference:** \(\Gamma; \Psi \vdash t \uparrow \leadsto \Psi' \vdash t' : T\)
- **Checking:** \(\Gamma; \Psi \vdash t \downarrow T \leadsto \Psi' \vdash t' : T\)

\text{Pretyping.pretype, Typing.infer, Typing.check}

\[
\text{CHECK-TYPE} \\
\theta \vdash \text{Type}_{i+1} \leq T \leadsto \theta' \\
\Gamma; \text{us} \vdash \theta \vdash \text{Type} \downarrow T \leadsto \text{us, } i \vdash \theta' \vdash \text{Type}_i : T
\]
Suppose a top-level \texttt{Definition id : }T := t.
Suppose a top-level `Definition id : T := t.`

1 \[ \vdash T \uparrow \rightsquigarrow \Psi \vdash T' : s \text{ (pretype)} \]
Introducing universe polymorphic definitions

Suppose a top-level `Definition id : T := t.`

1. \[ \vdash T \uparrow \rightsquigarrow \Psi \vdash T' : s \text{ (pretype)} \]
2. \[ \Psi \vdash t \downarrow T' \rightsquigarrow ; i \models \theta \vdash t : T' \text{ (infer_conv)} \]
Introducing universe polymorphic definitions

Suppose a top-level \textbf{Definition} \texttt{id : T := t}.

1. \(\vdash T \uparrow \rightsquigarrow \Psi \vdash T' : s\) (pretype)
2. \(\Psi \vdash t \Downarrow T' \rightsquigarrow; i \models \theta \vdash t : T'\) (infer\_conv)
3. Add \texttt{id : \forall i \models \theta, T' := t} to the environment (Typeops.infer, Reduction.conv).
4. If monomorphic: Add \(i \models \Theta\) to the global universe environment and \texttt{id : T' := t} separately. (Environ.push\_context).

Guiding principle:
Constants are transparent, \textcolor{red}{indistinguishable} from their bodies.

Global vs local universes: \(i\) is global \(\Rightarrow i > \texttt{Set}\), otherwise \(i \geq \texttt{Set}\).
Using universe polymorphic definitions

\[
\text{Infer-Cst}
\]

\[
\frac{(\text{id} : \forall i \models \theta, T) \in \Sigma \quad \vec{l} \not\in \vec{u}}{
\Gamma; \vec{u} \models \Theta \models \text{id} \uparrow \leadsto \psi \models \text{id}_\vec{l} : T[\vec{l}/\vec{i}]}
\]

where \( \psi = \vec{u}, \vec{l} \models \Theta \cup \theta[\vec{l}/\vec{i}] \)

\(\Rightarrow\) Constants now carry their universe substitution-instance.

\(\Rightarrow\) Inductives and constructors treated the same way.

type Level = Prop | Set
| Level of int * DirPath.t (* global *)
| Var of int (* local, de Bruijn index *)

type Univ.Instance.t = Level.t array

type 'a puniverses = 'a * Univ.Instance.t
Terms

```ocaml

type pconstant = constant puniverses
type pinductive = inductive puniverses
type pconstructor = constructor puniverses

type constr = ...
  | Sort       of Sorts.t
  | Const      of pconstant
  | Ind        of pinductive
  | Construct  of pconstructor
```

Matthieu Sozeau - Universe Polymorphism in Coq, for the OCaml hacker
Conversion

**Cumul-Sort**

\[ \psi \vdash i \, R \, j \]

\[ \text{Type}_i \overset{R}{=} \psi \text{ Type}_j \]

**Cumul-Prod**

\[ U \overset{=}{} \psi \, U' \quad T \overset{R}{=} \psi \, T' \]

\[ \Pi x : U.T \overset{R}{=} \psi \Pi x : U'.T' \]
Conversion

\[ \psi \vdash i \ R \ j \]
\[ \text{Type}_i =_\psi \text{Type}_j \]

\[ \text{Cumul-Sort} \]

\[ U \equiv_\psi U' \quad T \equiv_\psi T' \]
\[ \Pi x : U.T \equiv_\psi \Pi x : U'.T' \]

\[ \text{Cumul-Prod} \]

\[ \text{Conv-FQ} \]
\[ \overrightarrow{aS} \equiv_\psi \overrightarrow{bs} \]
\[ \psi \models \overrightarrow{u} = \overrightarrow{v} \]
\[ \overrightarrow{c u} \overrightarrow{aS} =_\psi \overrightarrow{c v} \overrightarrow{bs} \]

Uses backtracking (Reduction.conv)
Universes in Coq

1 Introduction

2 Elaborating Universes
   - Kernel
   - Engine
   - Unification
   - Minimization
When elaborating terms or proofs, the inferred universe context (evar_universe_context, UState.t) is part of the evar_map.

Evd.from_env : Global.env -> evar_map

(* Gensym *)
new_univ_level_variable : ?name:string -> rigid -> evar_map -> evar_map * Univ.Level.t

(* Adding constraints *)
Evd.set_leq_sort : env -> evar_map -> sorts -> sorts -> evar_map
Use two kinds of universe level variables during elaboration:

- Polymorphic constants get elaborated with fresh flexible argument levels by default.
- Typical ambiguity (e.g. `Type`) creates rigid variables.
- User-given levels (e.g. `Type@{i}`, `foo@{i}`) are rigid.

```
type rigid =  
  | UnivRigid
  | UnivFlexible of bool (* can be algebraic? *)

Evd.fresh_global : ?rigid:rigid -> env -> evar_map ->
                      global_reference -> evar_map * constr

(* For tactics *)
pf_constr_of_global : global_reference ->
                      (constr -> unit tactic) -> unit tactic
```
Unification of $id_i$ and $id_j$:

Definition $U_2 := \text{Type}_i$.
Definition $U_1 : U_2 := \text{Type}_j \rightsquigarrow j < i$
Definition $U_0 : U_1 := \text{Type}_k \rightsquigarrow k < j$
Definition $U_{02} : U_2 := U_0 \rightsquigarrow k < i$

\[
\begin{align*}
  id_j \ U_{02} & \sim id_i \ U_0 \rightsquigarrow i = j \\
\end{align*}
\]

But:

\[
\begin{align*}
  id_j \ U_{02} \to^* (U_0 \to U_0) \leftarrow^* id_i \ U_0
\end{align*}
\]

Unification also backtracks to ensure most general typings. 
**New**: also backtrack on unifications that would introduce inconsistencies (used to be found at Qed time only).
Unification with universes

\[ t \equiv^R_{\psi} u \leadsto \psi' \] unification of \( t \) and \( u \) under \( \psi \).

\[
\begin{align*}
\text{Elab-R-FO} & \quad \overrightarrow{a}s \equiv^\psi \overrightarrow{b}s \leadsto \psi' \quad \psi' \models \overrightarrow{u} \equiv \overrightarrow{v} \leadsto \psi'' \\
\text{c-} & \quad \overrightarrow{c}u \equiv^R \overrightarrow{a}s \equiv^\psi \overrightarrow{c}v \equiv \overrightarrow{b}s \leadsto \psi'
\end{align*}
\]
Unification with universes

\( t \equiv^R_\psi u \leadsto \psi' \): unification of \( t \) and \( u \) under \( \psi \).

\[
\begin{align*}
\text{Elab-R-FO} & \\
\overrightarrow{a} s \equiv_\psi \overrightarrow{b} s & \leadsto \psi' \\
\psi' & \models \overrightarrow{u} \equiv \overrightarrow{v} \leadsto \psi'' \\
\overleftarrow{c} \overrightarrow{u} \overrightarrow{a} s & \equiv^R \overleftarrow{c} \overrightarrow{b} s & \overleftarrow{c} \overrightarrow{u} \overrightarrow{a} s & \equiv_\psi \overleftarrow{c} \overrightarrow{b} s & \leadsto \psi'
\end{align*}
\]

\( \psi \models i \equiv j \leadsto \psi' \): unification of universe instances.

\[
\begin{align*}
\text{Elab-Univ-Eq} & \\
\psi \models i = j & \models i \equiv j \leadsto \psi
\end{align*}
\]

\[
\begin{align*}
\text{Elab-Univ-Flexible} & \\
i_f \lor j_f & \in \overrightarrow{u} s \\
\psi \land i = j & \models \models i \equiv j \leadsto \psi \land i = j
\end{align*}
\]
Levels vs Algebraics again

Universe instances are levels: Suppose

\[ \text{id} : \forall i \models \Pi A : \text{Type}_i, A \rightarrow A \]

\[ \Gamma = A : \text{Type}_i, P : \text{fibration}_{i,j} A \vdash \Sigma_{i,j} A P : \text{Type}_{\max(i,j)} \]

Levels only, adding constraint if an algebraic would appear:

\[ \Gamma; \vec{u} \models \theta \vdash \text{id} (\Sigma A P) \uparrow \vec{u}, k \models \theta \cup \max(i,j) \leq k \vdash \text{id}_k (\Sigma_{i,j} A P) \ldots \]
Universe instances are levels: Suppose

\[ \text{id} : \forall i \models, \Pi A : \text{Type}_i, A \rightarrow A \]

\[ \Gamma = A : \text{Type}_i, P : \text{fibration}_{i,j} A \vdash \Sigma_{i,j} A P : \text{Type}_{\max(i,j)} \]

Levels only, adding constraint if an algebraic would appear:

\[ \Gamma; \vec{u} \models \theta \vdash \text{id} (\Sigma A P) \uparrow \vec{u}, k \models \theta \cup \text{max}(i, j) \leq k \vdash \text{id}_k (\Sigma_{i,j} A P) \ldots \]
Minimization

That’s a lot of fresh universe variables!!

Typical example:

\[ \Gamma; \vdash \text{id true} \uparrow \leadsto i_f \vdash \text{Set} \leq i \vdash \@\text{id}_i \text{ bool true} : \text{bool} \]
Minimization

That’s a lot of fresh universe variables!!

Typical example:

\[ \Gamma; \vdash \text{id } \text{true } \uparrow \rightsquigarrow i \vdash \text{Set} \leq i \vdash @\text{id}_i \text{ bool true : bool} \]

We’d want: \( @\text{id}_{\text{Set}} \text{ bool true : bool} \), no new universe, no additional constraint, just as general as conversion will unfold \( \text{id}_{\text{Set}} \) if necessary.
That’s a lot of fresh universe variables!!

Typical example:

\[ \Gamma; \vdash \text{id } \text{true} \uparrow \leadsto i_f \models \text{Set} \leq i \vdash @i \text{id}_i \text{bool true : bool} \]

We’d want: \( @\text{id}_{\text{Set}} \text{bool true : bool} \), no new universe, no additional constraint, just as general as conversion will unfold \( i_{\text{id}}_{\text{Set}} \) if necessary.

\[ \Rightarrow \] Minimization: compute a minimal set of universe variables.

See Cardelli’s greedy algorithm for \( F^{\leq} \) inference, local type inference (Pierce & Turner).

- Only applies to flexible variables.
Normalization of universes

Before putting a definition/proof term into the environment:

\[
\text{Evd.nf}\_\text{constraints} : \text{evar}\_\text{map} \to \text{evar}\_\text{map}
\]

\[
\text{Evarutil.nf}\_\text{evars}\_\text{universes} :
  \text{evar}\_\text{map} \to \text{constr} \to \text{constr}
\]

\[
\text{Evd.universe}\_\text{context} : \text{?names} \to \text{evar}\_\text{map} \to
  (\text{Id.t} \ast \text{Level.t}) \text{list} \ast \text{Univ.universe}\_\text{context}
\]
Which comparison function to use? (e.g. for change, Ltac pattern-matching, ...) 

- Syntactic equality: eq_constr_nounivs, eq_constr_univs, eq_constr_univs_infer
- Conversion: Reductionops.check_conv, infer_conv
- Unification: evar_conv_x (no choice here)

We chose to use infer versions most of the time, assuming universe unifications are wanted. This required fixing threadings of the evar_map.
Due to obligation to register levels and constraints in the evar_map, and as global_references are no longer well-formed constrs (except monomorphic ones):

- Tactics should bind lazy global_references instead of lazy constrs.
- Term.eq_constr should be rare in tactics, many cases where Globnames.is_global should be used instead.
- Tactics need to ensure the terms they produce can be typed in the evar_map (e.g. with sufficient universe constraints). Otherwise use checked versions (e.g. exact_check) that do typechecking to ensure the constraints are inferred.
Universes must be declared before they are used:

- Problem with side-effect e.g. of Require Import during a proof. Must explicitly update the evar_map of the proof with the new constraints. Would be fixed by correctly threading the env in proof mode, with side effecting commands emitting their effects in that env.

- In tactics, any evar_map threading error can result in an anomaly.
Future plans

The main issue is the large number of universes and constraints generated (100's for a single definition).

- Cumulativity going through inductives (A. Timany), and definitions.

- Try to classify argument universes as inputs and outputs (syntactic check), and treat inputs like “template” polymorphic universes, not recording them. Loses compositionality: must check complete applications of polymorphic references.

- More algebraic universes, less constraints. Algebraics need heuristics in unification: $\max(i, j) = \max(k, l)$? (Agda, Lean have incomplete solutions). If one keeps non-normalized $\max(\_)$ universes, we can maybe avoid heuristics but make $\max(\_)$ expressions grow a lot.
At the end of elaboration: \( \overrightarrow{i} \models \Theta \vdash t : T \), with \( \theta \) a satisfiable set of constraints.

Find a minimal set of universes variables \( \overrightarrow{i} \subset \overrightarrow{i} \), universes \( \overrightarrow{u} \), a substitution \( \sigma : \overrightarrow{i} \rightarrow \overrightarrow{u} \) and constraints \( \Theta' \) s.t. \( \overrightarrow{i} \models \Theta' \cup \Theta\sigma \) and \( \overrightarrow{i} \models \Theta\sigma \Rightarrow \Theta' \).

- First normalize the constraints w.r.t. loops \( l \leq r \land r \leq l \) and equalities.
At the end of elaboration: \( \vec{i} \models \Theta \vdash t : T \), with \( \theta \) a satisfiable set of constraints.
Find a minimal set of universes variables \( \vec{i}' \subset \vec{i} \), universes \( \vec{u} \), a substitution \( \sigma : \vec{i} \rightarrow \vec{u} \) and constraints \( \Theta' \) s.t. \( \vec{i}' \models \Theta' \cup \Theta \sigma \) and \( \vec{i}' \models \Theta \sigma \Rightarrow \Theta' \).

- First normalize the constraints w.r.t. loops \((l \leq r \wedge r \leq l)\) and equalities.
- Canonicalize \( \Theta \) w.r.t equalities (except globals)
- Consider the remaining undefined flexible universe variables.
We now have $\Theta$ with only inequality constraints and a set $f$ of flexible universe variables.

- Let $i \in f$, compute its g.l.b: $\{\max(\vec{j}), j \ O \ i \in \Theta\}$. If $i$ has no lower constraints it must be kept.
- Generate upper constraints $\{\text{glb} \ O \ j \mid i \ O \ j \in \Theta\}$
- Set $i := \text{glb}$ except if $\text{glb}$ is algebraic and $i$ has upper constraints.
The End