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?

Universes are the types of *types*, e.g:

- nat, bool : $\texttt{Type}_0$
- $\texttt{Type}_0$: $\texttt{Type}_1$
- list : $\texttt{Type}_0 \to \texttt{Type}_0$
- $\forall \alpha : \texttt{Type}_0,$ list $\alpha : \texttt{Type}_1$
- $\forall n :$ nat$, \{n = 0\} + \{n \neq 0\} : \texttt{Type}_0$

How are they organised?

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

- Avoids the $\texttt{Type} : \texttt{Type}$ paradox (system U^-)
- Replicates RUSSELL's paradox of $\{x \mid x \notin x\}$, the set of all sets etc....
- Think of $\texttt{Type}_0$ as sets, $\texttt{Type}_1$ as classes etc...

COQ's theory

sort of t = type of the type of t, necessarily a $\texttt{Type}_i$.

$$\frac{\vdash \Gamma \quad (i \in \mathbb{N})}{\Gamma \vdash \texttt{Type}_i : \texttt{Type}_{i+1}}\ \textsc{Type-intro}$$

$$\frac{\Gamma \vdash A : \texttt{Type}_i \qquad \Gamma, x : A \vdash B : \texttt{Type}_j}{\Gamma \vdash \Pi x : A.B : \texttt{Type}_{\max(i,j)}}\ \textsc{Type-prod}$$

```
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.

$\Rightarrow$ 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 $\texttt{Type}_i$s.
But in general many i's can work!

$$\texttt{Definition}\ \mathrm{id}\ (A : \texttt{Type})\ (a : A) := a.$$

$$\rightsquigarrow\ \vdash \mathrm{id} : \Pi(A : \texttt{Type}_0),\ A \to A : \texttt{Type}_1$$

or

$$\rightsquigarrow\ \vdash \mathrm{id} : \Pi(A : \texttt{Type}_1),\ A \to A : \texttt{Type}_2$$

or . . . ?

$\Rightarrow$ universe variables

```
type Level.t = Prop | Set
     | Level of int * DirPath.t (* global *)
```

Floating universes and constraints

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

Type-intro

$$\frac{\vdash_{float} \Gamma \quad (i, j \in \mathbb{L})}{\Gamma \vdash_{float} \texttt{Type}_i : \texttt{Type}_j \leadsto i < j}$$

Type-prod

$$\frac{\Gamma \vdash_{float} A : \texttt{Type}_i \qquad \Gamma, x : A \vdash B : \texttt{Type}_j}{\Gamma \vdash_{float} \Pi x : A.B : \texttt{Type}_k \leadsto \max(i, j) \leq k}$$

```
type constraint_type = Lt | Le | Eq
type univ_constraint = Level.t * constraint_type * Level.t
module Contraint.t : Set.S with type elt = univ_constraint
```

Algebraic vs Atomic Constraints

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.

Algebraic vs Atomic Constraints

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:

$$\texttt{Definition}\ \mathrm{id}\ (A : \texttt{Type})\ (a : A) := a$$

$$\leadsto\ \vdash \mathrm{id} : \Pi(A : \texttt{Type}_l),\ A \to A : \texttt{Type}_{l+1}$$

$\Rightarrow$ l is *not* quantified at the definition level here, it is *global*:

$$\not\vdash \mathrm{id}\ (\Pi(A : \texttt{Type}_l),\ A \to A)\ \mathrm{id} : \tau$$

Because $l + 1 \not\leq l$. However l can gradually move up as high as wanted.

With polymorphism

Bounded polymorphism:

$$\texttt{Polymorphic Definition}\ \mathrm{id}\ (A : \texttt{Type})\ (a : A) := a$$

$$\underline{\mathrm{id}}_l : \Pi(A : \texttt{Type}_l),\ A \to A$$

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

$$l < k \vdash_{poly} \underline{\mathrm{id}}_k\ (\Pi(A : \texttt{Type}_l),\ A \to A)\ \underline{\mathrm{id}}_l : (\Pi(A : \texttt{Type}_l),\ A \to A)$$

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	Ψ	$::=$	$\overrightarrow{i} \models \Theta$
`Univ.UContext.t`	$=$		`Level.t array` $*$ `constraints`
`Univ.ContextSet.t`	$=$		`LSet.t` $*$ `constraints`

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	Ψ	$::=$	$\overrightarrow{i} \models \Theta$
`Univ.UContext.t`	$=$		`Level.t array` $*$ `constraints`
`Univ.ContextSet.t`	$=$		`LSet.t` $*$ `constraints`

Elaboration in bidirectionl 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.

universe context	Ψ	$::=$	$\vec{i} \vDash \Theta$
`Univ.UContext.t`	$=$		`Level.t array` $*$ `constraints`
`Univ.ContextSet.t`	$=$		`LSet.t` $*$ `constraints`

Elaboration in bidirectionl 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`

Check-Type

$$\frac{\theta \vdash \texttt{Type}_{i+1} \leq T \rightsquigarrow \theta'}{\Gamma; us \vDash \theta \vdash \texttt{Type} \Downarrow T \rightsquigarrow us, i \vDash \theta' \vdash \texttt{Type}_i : T}$$

Introducing universe polymorphic definitions

Suppose a top-level `Definition` $\mathrm{id} : T := t$.

Introducing universe polymorphic definitions

Suppose a top-level `Definition` $\mathrm{id} : T := t$.

1. $; \vdash T \Uparrow \rightsquigarrow \Psi \vdash T' : s$ (`pretype`)

Introducing universe polymorphic definitions

Suppose a top-level `Definition` 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`)

Introducing universe polymorphic definitions

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

1. $; \vdash T \Uparrow \rightsquigarrow \Psi \vdash T' : s$ (`pretype`)
2. $; \Psi \vdash t \Downarrow T' \rightsquigarrow ; i \vDash \theta \vdash t : T'$ (`infer_conv`)
3. Add id : $\forall\ i \vDash \theta, T' := t$ to the environment (`Typeops.infer, Reduction.conv`).
4. If monomorphic: Add $i \vDash \Theta$ to the global universe environment and id : $T' := t$ separately. (`Environ.push_context`).

Guiding principle:
Constants are transparent, indistinguishable from their bodies.

Global vs local universes: i is global $\Rightarrow i >$ `Set`, otherwise $i \geq$ `Set`.

Using universe polymorphic definitions

$$\frac{(\mathrm{id} : \forall\, i \vDash \theta, T) \in \Sigma \qquad \overrightarrow{l} \notin \overrightarrow{u}}{\Gamma; \overrightarrow{u} \vDash \Theta \vdash \mathrm{id} \Uparrow \rightsquigarrow \psi \vdash \mathrm{id}_{\overrightarrow{l}} : T[\overrightarrow{l} / \overrightarrow{i}]}\ \textsc{Infer-Cst}$$

where $\psi = \overrightarrow{u}, \overrightarrow{l} \vDash \Theta \cup \theta[\overrightarrow{l} / \overrightarrow{i}]$

⇒ Constants now carry their universe substitution/instance.

⇒ 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

```
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
```

Conversion

Cumul-Sort

$$\frac{\psi \models i \; R \; j}{\texttt{Type}_i =_{\psi}^{R} \texttt{Type}_j}$$

Cumul-Prod

$$\frac{U =_{\psi}^{=} U' \qquad T =_{\psi}^{R} T'}{\Pi x : U.T =_{\psi}^{R} \Pi x : U'.T'}$$

Conversion

Cumul-Sort

$$\frac{\psi \models i \; R \; j}{\texttt{Type}_i =_\psi^R \texttt{Type}_j}$$

Cumul-Prod

$$\frac{U =_\psi^= U' \qquad T =_\psi^R T'}{\Pi x : U.T =_\psi^R \Pi x : U'.T'}$$

Conv-FO

$$\frac{\overrightarrow{as} =_\psi^= \overrightarrow{bs} \qquad \psi \models \overrightarrow{u} = \overrightarrow{v}}{\underline{c}_{\overrightarrow{u}} \; \overrightarrow{as} =_\psi^R \underline{c}_{\overrightarrow{v}} \; \overrightarrow{bs}}$$

Uses backtracking (`Reduction.conv`)

Universe contexts

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
```

Levels

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

Unification of id_i and id_j:

$\texttt{Definition}\ U2 := \texttt{Type}_i.$

$\texttt{Definition}\ U1 : U2 := \texttt{Type}_j \leadsto j < i$

$\texttt{Definition}\ U0 : U1 := \texttt{Type}_k \leadsto k < j$

$\texttt{Definition}\ U02 : U2 := U0 \leadsto k < i$

$$\mathrm{id}_j\ U02\ \sim\ \mathrm{id}_i\ U0 \leadsto i = j$$

But:

$$\mathrm{id}_j\ U02 \rightarrow^* (U0 \rightarrow U0) \leftarrow^* \mathrm{id}_i\ U0$$

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 ψ.

$$\textsc{Elab-R-FO}\quad \frac{\overrightarrow{as} \equiv^{=}_{\psi} \overrightarrow{bs} \leadsto \psi' \qquad \psi' \models \overrightarrow{u} \equiv \overrightarrow{v} \leadsto \psi''}{\underline{\mathrm{c}}_{\overrightarrow{u}}\ \overrightarrow{as} \equiv^{\mathsf{R}}_{\psi} \underline{\mathrm{c}}_{\overrightarrow{v}}\ \overrightarrow{bs} \leadsto \psi'}$$

Unification with universes

$t \equiv^R_\psi u \rightsquigarrow \psi'$: unification of t and u under ψ.

$$\textsc{Elab-R-FO}\quad \frac{\overrightarrow{as} \equiv^{=}_{\psi} \overrightarrow{bs} \rightsquigarrow \psi' \qquad \psi' \models \overrightarrow{u} \equiv \overrightarrow{v} \rightsquigarrow \psi''}{\underline{c}_{\overrightarrow{u}}\ \overrightarrow{as} \equiv^{R}_{\psi} \underline{c}_{\overrightarrow{v}}\ \overrightarrow{bs} \rightsquigarrow \psi'}$$

$\psi \models i \equiv j \rightsquigarrow \psi'$: unification of universe instances.

$$\textsc{Elab-Univ-Eq}\quad \frac{\psi \models i = j}{\psi \models i \equiv j \rightsquigarrow \psi}$$

$$\textsc{Elab-Univ-Flexible}\quad \frac{i_{\mathsf{f}} \vee j_{\mathsf{f}} \in \overrightarrow{u_s} \qquad \psi \wedge i = j \models}{(\overrightarrow{u_s} \models \psi) \models i \equiv j \rightsquigarrow \psi \wedge i = j}$$

Levels vs Algebraics again

Universe instances are levels: Suppose

$$\mathrm{id} : \forall i \vDash, \Pi A : \mathtt{Type}_i, A \to A$$

$$\Gamma = A : \mathtt{Type}_i, P : \mathrm{fibration}_{i,j} A \vdash \Sigma_{ij}\ A\ P : \mathtt{Type}_{\mathsf{max}(i,j)}$$

Levels only, adding constraint if an algebraic would appear:

$$\Gamma; \overrightarrow{u} \vDash \theta \vdash \mathrm{id}\ (\Sigma\ A\ P) \Uparrow \overrightarrow{u}, k \vDash \theta \cup \mathsf{max}(i,j) \leq k \vdash \mathrm{id}_k(\Sigma_{ij}\ A\ P) \ldots$$

Levels vs Algebraics again

Universe instances are levels: Suppose

$$\mathrm{id} : \forall i \vDash, \Pi A : \mathtt{Type}_i, A \to A$$

$$\Gamma = A : \mathtt{Type}_i, P : \mathrm{fibration}_{i,j} A \vdash \Sigma_{ij}\ A\ P : \mathtt{Type}_{\mathsf{max}(i,j)}$$

Levels only, adding constraint if an algebraic would appear:

$$\Gamma; \overrightarrow{u} \vDash \theta \vdash \mathrm{id}\ (\Sigma\ A\ P) \Uparrow \overrightarrow{u}, k \vDash \theta \cup \mathsf{max}(i,j) \leq k \vdash \mathrm{id}_k(\Sigma_{ij}\ A\ P) \ldots$$

Minimization

That's *a lot* of fresh universe variables!!

Typical example:

$$\Gamma; \vdash \mathrm{id}\ \mathtt{true} \Uparrow \rightsquigarrow i_{\mathsf{f}} \models \mathtt{Set} \leq i \vdash @\mathrm{id}_i\ \mathsf{bool}\ \mathsf{true} : \mathsf{bool}$$

Minimization

That's *a lot* of fresh universe variables!!

Typical example:

$$\Gamma; \vdash \mathrm{id}\ \mathsf{true} \Uparrow \rightsquigarrow i_{\mathsf{f}} \vDash \mathtt{Set} \leq i \vdash @\mathrm{id}_i\ \mathsf{bool}\ \mathsf{true} : \mathsf{bool}$$

We'd want: $@\mathrm{id}_{\mathtt{Set}}\ \mathsf{bool}\ \mathsf{true} : \mathsf{bool}$, no new universe, no additional constraint, just as general as conversion will unfold $\mathrm{id}_{\mathtt{Set}}$ if necessary.

Minimization

That's *a lot* of fresh universe variables!!

Typical example:

$$\Gamma; \vdash \mathrm{id}\ \mathsf{true} \Uparrow \rightsquigarrow i_{\mathsf{f}} \vDash \mathtt{Set} \leq i \vdash @\mathrm{id}_i\ \mathsf{bool}\ \mathsf{true} : \mathsf{bool}$$

We'd want: $@\mathrm{id}_{\mathtt{Set}}$ bool true : bool, no new universe, no additional constraint, just as general as conversion will unfold $\mathrm{id}_{\mathtt{Set}}$ 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:

```
Evd.nf_constraints : evar_map -> evar_map

Evarutil.nf_evars_universes :
  evar_map -> constr -> constr

Evd.universe_context : ?names -> evar_map ->
  (Id.t * Level.t) list * Univ.universe_context
```

Design/implementation issues

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`.

Design/implementation issues

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.

Known Bugs

Universes must be declared before they are used:

- Problem with side-effect e.g. of `Require Import` during a proof. Must explicitely 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. Looses 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.

Minimization II: `Universes.normalize_context_set`

At the end of elaboration: $\overrightarrow{i} \vDash \Theta \vdash t : T$, with θ a satisfiable set of constraints.
Find a mimimal set of universes variables $\overrightarrow{i'} \subset \overrightarrow{i}$, universes $\overrightarrow{u}$, a substitution $\sigma : \overrightarrow{i} \to \overrightarrow{u}$ and constraints Θ' s.t. $\overrightarrow{i'} \vDash \Theta' \cup \Theta\sigma$ and $\overrightarrow{i'} \vDash \Theta\sigma \Rightarrow \Theta'$.

- First normalize the constraints w.r.t. loops ($l \leq r \wedge r \leq l$) and equalities.

At the end of elaboration: $\overrightarrow{i} \vDash \Theta \vdash t : T$, with θ a satisfiable set of constraints.
Find a mimimal set of universes variables $\overrightarrow{i'} \subset \overrightarrow{i}$, universes $\overrightarrow{u}$, a substitution $\sigma : \overrightarrow{i} \rightarrow \overrightarrow{u}$ and constraints Θ' s.t. $\overrightarrow{i'} \vDash \Theta' \cup \Theta\sigma$ and $\overrightarrow{i'} \vDash \Theta\sigma \Rightarrow \Theta'$.

- First normalize the constraints w.r.t. loops ($l \leq r \wedge r \leq l$) and equalities.
- Canonicalize Θ w.r.t equalities (except globals)
- Consider the remaining undefined flexible universe variables.

Minimization III

We now have Θ with only inequality constraints and a set f of flexible universe variables.

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

The End