Universe Polymorphism: Subtyping and Unification

Matthieu Sozeau

TYPEX Days
December 17th 2013
Paris

Project Team $\pi r^2$
INRIA & PPS, Paris 7
What are universes?

Universes are the types of types, e.g:

- nat, bool : $\text{Type}_0$
- $\text{Type}_0 : \text{Type}_1$
- list : $\text{Type}_0 \rightarrow \text{Type}_0$
- $\forall \alpha : \text{Type}_0, \text{list} \alpha : \text{Type}_1$
- $\forall n : \text{nat}, \{n = 0\} + \{n \neq 0\} : \text{Type}_0$
A hierarchy of universes $\text{Type}_0 < \text{Type}_1 < \ldots \text{Type}_n$ used to avoid the Type : Type paradox (system $U^-$, first version of Martin-Löf Type Theory (MLTT)).

- Replicates Russell’s paradox of $\{x \mid x \notin x\}$, the set of all sets etc....
- You can think of $\text{Type}_0$ as sets, $\text{Type}_1$ as classes etc...
We call the sort of $t$ the type of the type of $t$, it is necessarily a $\text{Type}_i$ (see e.g. \textsc{Type-prod}).

\[
\text{Type-intro} \quad 
\begin{array}{c}
\vdash \Gamma \ (i \in \mathbb{N}) \\
\Gamma \vdash \text{Type}_i : \text{Type}_{i+1}
\end{array}
\]

\[
\text{Type-prod} \quad 
\begin{array}{c}
\Gamma \vdash A : \text{Type}_i \\
\Gamma, x : A \vdash B : \text{Type}_j
\end{array} \\
\Gamma \vdash \Pi x : A. B : \text{Type}_{\max(i,j)}
\]

For our purposes, think of Coq’s type theory as a stratified, predicative System F + primitive inductive/algebraic datatypes. The dependent quantification on terms is not relevant here.
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 works (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.

In practice, translation from a source language with anonymous Types to a core theory with Type_'s only.
Floating universes

But in general many \( i \)'s can work!

**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 \ldots

\[\Rightarrow \text{Allow universe variables.}\]
Consistency is now decided by giving an assignation of natural numbers to universe variables, satisfying constraints. New judgment $\vdash_{float}$

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

\[
\text{TYPE-PROD} \\
\Gamma \vdash_{float} A : \text{Type}_{i} \quad \Gamma, x : A \vdash B : \text{Type}_{j} \\
\Gamma \vdash_{float} \Pi x : A.\text{B} : \text{Type}_{k} \leadsto \max(i, j) \leq k
\]
Proposition (Correctness)

If $\Gamma \vdash_{\text{float}} t : T \rightsquigarrow \Theta$ and $\Theta$ is satisfiable (with assignment $\sigma$) then $\Gamma[\sigma] \vdash t[\sigma] : T[\sigma]$.

Constraints $\Rightarrow$ graph structure based on union-find. $l < l'$ is an arc, consistency is cycle-freeness, easy to check incrementally.
Floating levels give a false sense of polymorphism:

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

\[
\vdash \text{id} : \Pi(A : \text{Type}_l), \ A \to A : \text{Type}_{\text{max}(l+1,l)} \equiv \text{Type}_{l+1}
\]

\[
\Rightarrow l \text{ is not quantified at the definition level here, it is } \text{global}: 
\]

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

Because \( l + 1 \not\leq l \).
With polymorphism

Real, bounded polymorphism:

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

\[ \text{id} : \forall \ l, \Pi(A : \text{Type}_l), \ A \to A \]

\( \Rightarrow l \text{ is quantified at the definition level now, but second-class, we need } \text{instantiation}: \)

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

2 The current setup
   - Definitions
   - Issues

3 The new setup
   - Universe polymorphic definitions
   - The good, the bad and the ugly
   - Minimizing the ugly
   - Dealing with Prop
   - Implementation & benchmarks

4 The past & the future
Implicit universes with cumulativity, à la Russell.

*In the kernel*, build up a set of universe constraints $\Theta$.

\[
\text{PROD} \quad \Gamma; \Theta \vdash T : \text{Type}_i \leadsto \Theta_1 \quad \Gamma, x : T; \Theta_1 \vdash U : \text{Type}_j \leadsto \Theta_2 \\
\quad \Gamma; \Theta \vdash \Pi x : T.U : \text{Type}_{\max(i,j)} \leadsto \Theta_2
\]

\[
\text{CONV} \quad \Gamma; \Theta \vdash t : U \leadsto \Theta_1 \\
\quad \Gamma; \Theta_1 \vdash V : s \leadsto \Theta_2 \quad \Theta_2 \vdash U \leq V \leadsto \Theta_3 \\
\quad \Gamma; \Theta \vdash t : V \leadsto \Theta_3
\]
**Kernel Conversion**

### Cumul-Sort

\[ \Theta \vdash \text{Type}_i \leq \text{Type}_j \leadsto \Theta \cup i \leq j \]

### Cumul-Prod

\[ \Theta \vdash U = U' \leadsto \Theta_1 \quad \Theta_1 \vdash T \leq T' \leadsto \Theta_2 \]

\[ \Theta \vdash \Pi x : U.T \leq \Pi x : U'.T' \leadsto \Theta_2 \]
Algebraic universes and constraints:

- **levels** \( i, j, le, lt \) \( \in \mathbb{N} \cup \{\text{Prop}, \text{Set}\} \)
- **universes** \( u, v \) \( ::= i \mid \max(le, lt) \)
- **successor** \( i + 1 \) \( ::= \max([], i) \)
- **order** \( O \) \( ::= = \mid < \mid \leq \)
- **atomic constraint** \( c \) \( ::= i O j \)
- **constraints** \( \Theta \) \( ::= \epsilon \mid c \cup \Theta \)

Only handles constraints of the form \( u O j \) by translation to atomic constraints:

\[
\max(i,j,k) \leq l \iff i \leq l \cup j \leq l \cup k < l
\]

Invariant on typing ensures this is the shape of inferred constraints (Herbelin, TYPES).
Some definitions

Algebraic universes and constraints:

levels \( i, j, le, lt \in \mathbb{N} \cup \{\text{Prop}, \text{Set}\} \)

universes \( u, v ::= i | \max(le, lt) \)

successor \( i + 1 ::= \max([], i) \)

order \( O ::= = | < | \leq \)

atomic constraint \( c ::= i O j \)

constraints \( \Theta ::= \epsilon | c \cup \Theta \)

Only handles constraints of the form \( u O j \) by translation to atomic constraints:

\[ \max(i, j, k) \leq l \iff i \leq l \cup j \leq l \cup k < l \]

Invariant on typing ensures this is the shape of inferred constraints (Herbelin, TYPES).
Issues

- Constraints are regenerated at each type checking
- Forces the global, unorderly generation of universe variables. Any term going out of the kernel must get refreshed universe variables because \( \max(i, j) \) shouldn’t be fed back to it.
- To implement universe polymorphism, must hack directly inside the kernel. Done for inductive types for now.
Universes in Coq

1. Introduction

2. The current setup
   - Definitions
   - Issues

3. The new setup
   - Universe polymorphic definitions
   - The good, the bad and the ugly
   - Minimizing the ugly
   - Dealing with Prop
   - Implementation & benchmarks

4. The past & the future
universe context \( \Psi \ ::= \overset{i}{\to} \vdash \Theta \)

Constraints are generated once at refinement time (outside the kernel):

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

**Checking:** \( \Gamma; \Psi \vdash t \downarrow T \rightsquigarrow \Psi' \vdash t' : T \)
universe context \( \Psi ::= \vec{i} \models \Theta \)

Constraints are generated once at refinement time (outside the kernel):

Inference: \( \Gamma; \Psi \vdash t \uparrow \leadsto \Psi' \vdash t' : T \)

Checking: \( \Gamma; \Psi \vdash t \downarrow T \leadsto \Psi' \vdash t' : T \)

\[
\text{CHECK-TYPE} \\
\frac{\theta \vdash \text{Type}_{i+1} \leq T \leadsto \theta'}{\Gamma; us \models \theta \vdash \text{Type} \downarrow T \leadsto us, i \models \theta' \vdash \text{Type}_{i} : T}
\]

\[
\text{INFER-CST} \\
\frac{(\text{id} : T) \in \Sigma}{\Gamma; \Psi \vdash \text{id} \uparrow \leadsto \Psi \vdash \text{id} : T}
\]
Constraint checking

universe context \( \Psi \ ::= \vec{i} \vdash \Theta \)

Constraints are generated once at refinement time (outside the kernel):
Inference: \( \Gamma; \Psi \vdash t \uparrow \leadsto \Psi' \vdash t' : T \)
Checking: \( \Gamma; \Psi \vdash t \downarrow T \leadsto \Psi' \vdash t' : T \)

**Check-Type**

\[
\begin{align*}
\theta \vdash \text{Type}_{i+1} \leq T & \leadsto \theta' \\
\Gamma; us \vdash \theta \vdash \text{Type} \downarrow T & \leadsto us, i \vdash \theta' \vdash \text{Type}_i : T
\end{align*}
\]

**Infer-Cst**

\[
\begin{align*}
(id : T) & \in \Sigma \\
\Gamma; \Psi \vdash id \uparrow \leadsto \Psi' \vdash id : T
\end{align*}
\]

- The kernel just checks constraints: \( \Gamma; \Psi \vdash t : T \)
- All universes and constraints that appear in the derivation
Now we can introduce universe polymorphism. Suppose a top-level definition $\text{id} := t : T$. 
Now we can introduce universe polymorphism. Suppose a top-level definition $\text{id} := t : T$. 

1. $\Gamma; \vdash T \uparrow \leadsto \Psi \vdash T' : s$
Now we can introduce universe polymorphism. Suppose a top-level definition \( \text{id} := t : T \). 

1. \( \Gamma; \vdash T \uparrow \leadsto \Psi \vdash T' : s \)

2. \( \Gamma; \Psi \vdash t \downarrow T' \leadsto i \models \theta \vdash t : T' \)
Now we can introduce universe polymorphism. Suppose a top-level definition \( \text{id} := t : T \).

1. \( \Gamma; \vdash T \overset{\uparrow}{\Rightarrow} \Psi \vdash T' : s \)
2. \( \Gamma; \Psi \vdash t \overset{\downarrow}{\Rightarrow} T' \sim \theta \vdash t : T' \)
3. Add \( \text{id} : \forall i \vdash \theta, T' := t \) to the environment.

\( \Rightarrow \) Guiding principle: constants are \textit{transparent}, indistinguishable from their bodies.
Universe Polymorphic definitions

To use $\text{id}$, we change elaboration of constants to:

$$
\text{ Infer-Cst }
\quad
\frac{(\text{id} : \forall i \models \theta, T) \in \Sigma}{\Gamma; \overrightarrow{u} \vdash \Theta \vdash \text{id} \uparrow \leadsto \overrightarrow{u}, i' \models \Theta \cup \theta[i' / i] \vdash \text{id}_{i'} : T[i' / i]}

\Rightarrow \quad \text{Constants now carry their universe substitution-instance.}
\Rightarrow \quad \text{Inductives and constructors treated the same way.}
Universe Polymorphic definitions: conversion

\[
\frac{c \rightarrow \delta \; t}{c \rightarrow \delta \; t} \quad \frac{t \rightarrow \delta = R \psi \; u}{t \rightarrow \delta = R \psi \; u}
\]

\[
\frac{c \rightarrow \delta \; t}{c \rightarrow \delta \; t} \quad \frac{t = R \psi \; u \rightarrow \delta}{t = R \psi \; u \rightarrow \delta}
\]

\[
\frac{R-\delta-FO}{\overrightarrow{a} \overrightarrow{s} = \overrightarrow{b} \overrightarrow{s}} \quad \frac{\psi \models \overrightarrow{u} = \overrightarrow{v}}{\psi \models \overrightarrow{u} = \overrightarrow{v}}
\]

\[
\frac{c \rightarrow \overrightarrow{a} \overrightarrow{s} = R \psi \; c \rightarrow \overrightarrow{b} \overrightarrow{s}}{c \rightarrow \overrightarrow{a} \overrightarrow{s} = R \psi \; c \rightarrow \overrightarrow{b} \overrightarrow{s}}
\]
Advantages to elaboration

- Reduced trusted code base: checking vs inference.
- Reduced polymorphism-specific code (actually no, thanks to backward compatibility).
- Avoid diffuse use of global gensym ⇒ more functional.
- User-level control on generated universes and form of constraints (simplification, declaration...).
- Mixing polymorphic and monomorphic definitions.
Disadvantage (for me and some of you): unification and tactics must become universe-aware.

Universes.constr_of_global :
  global_reference -> constr in_universe_context

Unification of $\text{id}_i$ and $\text{id}_j$: Syntactic equality of $i$ and $j$? Do nothing?
Due to (notoriously heuristic) first-order unification/conversion of constants... we could get too strict universe constraints.

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 $U_{02} : U_2 := U_0 \rightsquigarrow k < i$

\[ \text{id}_j U_{02} \sim \text{id}_i U_0 \rightsquigarrow i = j \]

But: $\text{id}_j U_{02} \rightarrow^* (U_0 \rightarrow U_0)$ and $\text{id}_i U_0 \rightarrow^* (U_0 \rightarrow U_0)$

$\Rightarrow$ Analyse variance of universes to relax first-order unification. Variance could help minimization too.

E.g. for $\text{id}_i t \sim \text{id}_j u$, $i$ and $j$ do not have to be compared.

$\Rightarrow$ But it looks like the analysis is hard, not compositional and requires full normalisation of the constant bodies.
Evd.fresh_global : \(\text{?rigid:rigid \rightarrow env \rightarrow evar\_map \rightarrow}
\text{global\_reference \rightarrow evar\_map \ast constr}\)

```haskell
type rigid =
  \_ UnivRigid
  \_ UnivFlexible of bool (* can be algebraic? *)
```

- Polymorphic constants get elaborated with flexible argument levels.
- Typical ambiguity (e.g. `Type`) creates rigid variables.
- User-given levels will be rigid
Unification with universes

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

\[
\begin{align*}
\text{Elab-R-} \delta \text{-left} & \quad \text{Elab-R-} \delta \text{-right} \\
\frac{c \rightarrow_{\iota} \rightarrow_{\delta} t \quad t \equiv^R_{\psi} u \leadsto \psi'}{c \rightarrow_{\iota} \overrightarrow{d} \equiv^R_{\psi} u \leadsto \psi'} & \quad \frac{c \rightarrow_{\iota} \rightarrow_{\delta} u \quad t \equiv^R_{\psi} u \overrightarrow{d} \leadsto \psi'}{t \equiv^R_{\psi} c \rightarrow_{\iota} \overrightarrow{d} \leadsto \psi'}
\end{align*}
\]

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

\[ \psi \models i \equiv j \leadsto \psi' : \text{unification of universe instances.} \]

\[
\begin{align*}
\text{Elab-Univ-EQ} & \quad \text{Elab-Univ-Flexible} \\
\frac{\psi \models i = j}{\psi \models i \equiv j \leadsto \psi} & \quad \frac{i_f \lor j_f \in u_s \quad \psi \land i = j \models}{(u_s \models \psi) \models i \equiv j \leadsto \psi \land i = j}
\end{align*}
\]
Universe instances are levels: Suppose

$$\text{id} : \forall i \vdash, \Pi A : \text{Type}_i, A \to A$$

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

Levels only, adding constraint if an algebraic would appear:

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

\[ \text{id} : \forall i \vdash, \Pi A : \text{Type}_i, A \to A \]

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

Levels only, adding constraint if an algebraic would appear:

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

That’s a lot of fresh universe variables!!

Typical example:

$$\Gamma; \Psi \vdash \text{id} \, \text{true} \uparrow \sim \Psi \cup i \models \text{Set} \leq i \vdash @\text{id}_i \, \text{bool} \, \text{true} : \text{bool}$$
Minimization

That’s a lot of fresh universe variables!!

Typical example:

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

We’d want: \( \text{@id}_{\text{Set}} \text{ bool true} : \text{bool} \), no new universe, no additional constraint, just as general.
Minimization

That’s a lot of fresh universe variables!!

Typical example:

$$\Gamma; \Psi \vdash \text{id} \ true \uparrow \leadsto \Psi \cup i \vdash \text{Set} \leq i \vdash @id_i \ 	ext{bool} \ true : \text{bool}$$

We’d want: $@id_{\text{Set}} \ 	ext{bool} \ true : \text{bool}$, no new universe, no additional constraint, just as general.

⇒ Minimization: compute a minimal set of universe variables.

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

▶ Requires no other lower constraints on $i$ ($j \ O \ i$).

▶ Only applies to flexible variables.
Minimization, results

Correctness proof: easy, preservation of local solutions.

Of course this is *not* endangering the consistency of Coq!

**Theorem (Conservativity)**

*Unfolding universe polymorphic definitions gives correct typings in the original system. Might just not be the most general ones if minimization did anything. For inductives, each instantiation is a new copy.*
Dealing with $\text{Prop}$

Let $\text{false}_i : \text{Type}_{i+1} \triangleq (\prod A : \text{Type}_i, A : \text{Type}_{\text{max}(i+1,i)})$.

But $\text{false}_\text{Prop} \rightarrow^* \prod A : \text{Prop}, A$, of type $\text{Prop}$ by impredicativity (and $\text{Type}_{\text{Prop}+1}$ still).
Let $\text{false}_i : \text{Type}_{i+1} \triangleq (\Pi A : \text{Type}_i, A : \text{Type}_{\max(i+1,i)})$.

But $\text{false}_{\text{Prop}} \rightarrow^* \Pi A : \text{Prop}, A$, of type $\text{Prop}$ by impredicativity (and $\text{Type}_{\text{Prop}+1}$ still).

**Fact:** our universe polymorphism is *incompatible* with the implicit use of impredicativity (which has computational content according to homotopy models, see hProp’s in the HoTT book). The implicit $\text{Prop} \leq \text{Type}$ rule also causes problems for models and syntax of proof-irrelevance (Werner *et al*)...  

**Ideal Solution:** Let’s get the Rooster and the Syntactic Bracket (Herbelin & Spiwack).
Dealing with $\text{Prop}$ II

Partial Solutions:

- Allow to disable $\text{Prop}$ completely, with $-\text{no-prop}$, for HoTTists.
- Disallow instantiating a parameter level $i$ with $\text{Prop}$, similar to the current Coq solution. But get less precise types.
- Currently, don’t mind the problem, it’s ok in practice and you don’t want to lose the benefit of:

$$\text{subset}_i \; (A : \text{Type}_i) \; (P : A \to \text{Prop}) : \text{Type}_i \triangleq (\Sigma_{i, \text{Prop}} A \; P : \text{Type}_{\text{max}(i, \text{Prop})})$$
Implementation still in progress but:

- Runs the Homotopy Type Theory Coq library with full universe polymorphism. No noticeable slowdown. Most definitions polymorphic on 6 universes at most.

- A universe polymorphic formalization of weak groupoids + an interpretation of CC in groupoids, takes 5 min to compile with polymorphism and fast projections, impossible without those two (inconsistency, exponential slowdown). 1min if deactivating universes.
Implementation

- Key technique: hash-consing for fast comparison of universe levels, universe instances and algebraic universes. Imperfect as deserialization breaks hashconsing (WIP with T. Braibant, PMP, ...).
- Minimization is fast.
- Currently using naive structures, i.e. the kernel side graph based on union-find does not do compression and the user-side (inference) substitution of universes does not use union-find. Optimize last!
- Universes have to be normalized at the end of inference now: `nf_evars_universes : evar_map -> constr -> constr`
1 Introduction

2 The current setup
   ■ Definitions
   ■ Issues

3 The new setup
   ■ Universe polymorphic definitions
   ■ The good, the bad and the ugly
   ■ Minimizing the ugly
   ■ Dealing with Prop
   ■ Implementation & benchmarks

4 The past & the future
Related work

- J. Courant: Explicit Universes for CC (TPHOLs’02). User-level declarations of $u \leq i$ in contexts, no other change.
- Matita (Coen et al.): checked universes, polymorphism at library level.
- Pierce and Turner (JFP): Local type inference (based on Cardelli’s greedy inference algorithm).
Nice things that become possible

- Universe polymorphic developments: reuse definitions and lemmas at different levels.
- Polymorphism for universes appearing \textit{inside} structures: old discrepancy between parameters and fields.
- Computational relations and rewriting: long standing limitation, e.g. for MathClasses. Useful for HoTT as well.
- Let us \textit{declare} universes and constraints (no user syntax yet).
- Resizing rules.
That’s all folks!
At the end of elaboration: \( \vec{i} \models \Theta \vdash t : T \).

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 \( (l \leq r \land r \leq l) \) and equalities.
At the end of elaboration: $\vec{i} \models \Theta \vdash t : T$.

Find a minimal set of universes variables $\vec{i}' \subseteq \vec{i}$, 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 ($l \leq r \land r \leq l$) and equalities.
- Canonicalize $\Theta$ w.r.t equalities (except globals)
- Mark $i$’s that are fresh universe variables from universe instances as candidates for unification + restriction for universes “on the left”.

Matthieu Sozeau - Universe Polymorphism: Subtyping and Unification 37
Minimization II

At the end of elaboration: $\vec{i} \models \Theta \vdash t : T$.
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 ($l \leq r \land r \leq l$) and equalities.
- Canonicalize $\Theta$ w.r.t equalities (except globals)
- Mark $i$’s that are fresh universe variables from universe instances as candidates for unification + restriction for universes “on the left”.
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: $\max(\overrightarrow{j}, j \in \Theta)$. 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 $\text{glb}$ algebraic and $i$ has upper constraints. We can share such $\text{glbs}$ though.