Universe Polymorphism: Subtyping and Unification

Matthieu Sozeau

Agda Implementers Meeting
May 27th 2014
Paris
Universes are the types of *types*, e.g:

- \( \text{nat}, \text{bool} : \text{Type}_0 \)
- \( \text{Type}_0 : \text{Type}_1 \)
- \( \text{list} : \text{Type}_0 \rightarrow \text{Type}_0 \)
- \( \forall \alpha : \text{Type}_0, \text{list} \ \alpha : \text{Type}_1 \)
- \( \forall n : \text{nat}, \{ n = 0 \} + \{ n \neq 0 \} : \text{Type}_0 \)
How are they organised?

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...
Coq’s theory

We call the sort of $t$ the type of the type of $t$, it is necessarily a $\text{Type}_i$ (see e.g. $\text{Type-prod}$).

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

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

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.
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 Typeᵢ'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 . . .

\[ \Rightarrow \text{Allow universe variables.} \]
Floating universes and constraints

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

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

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

If $\Gamma \vdash_{\text{float}} t : T \Rightarrow \Theta$ 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.
Without polymorphism

Floating levels give a false sense of polymorphism:

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

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

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

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

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

Real, bounded polymorphism:

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

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

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

\[ \vdash_{\text{poly}} \text{id}_k \ (\Pi(A : \text{Type}_l), \ A \rightarrow A) \ \text{id}_l : (\Pi(A : \text{Type}_l), \ A \rightarrow A) \rightsquigarrow k < l \]
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 $\Theta$.

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

\[
\text{CONV} \quad \Gamma; \Theta \vdash t : U \rightsimeq \Theta_1 \\
\Gamma; \Theta_1 \vdash V : s \rightsimeq \Theta_2 \quad \Theta_2 \vdash U \leq V \rightsimeq \Theta_3 \\
\Gamma; \Theta \vdash t : V \rightsimeq \Theta_3
\]
**Cumul-Sort**

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

**Cumul-Prod**

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

\[ \Theta \vdash \Pi x : U.T \leq \Pi x : U'.T' \rightsquigarrow \Theta_2 \]
### Some definitions

**Algebraic universes and constraints:**

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>levels</strong></td>
<td>$i, j, le, lt \in \mathbb{N} \cup {\text{Prop}, \text{Set}}$</td>
</tr>
<tr>
<td><strong>universes</strong></td>
<td>$u, v ::= i \mid \text{max}(le, lt)$</td>
</tr>
<tr>
<td><strong>successor</strong></td>
<td>$i + 1 ::= \text{max}([], i)$</td>
</tr>
<tr>
<td><strong>order</strong></td>
<td>$O ::= = \mid &lt; \mid \le$</td>
</tr>
<tr>
<td><strong>atomic constraint</strong></td>
<td>$c ::= i O j$</td>
</tr>
<tr>
<td><strong>constraints</strong></td>
<td>$\Theta ::= \epsilon \mid c \cup \Theta$</td>
</tr>
</tbody>
</table>

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**
  
  \[ u, v ::= i \mid \max(\text{le}, \text{lt}) \]

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

- **order**
  
  \[ \mathcal{O} ::= = \mid < \mid \leq \]

- **atomic constraint**
  
  \[ c ::= i \mathcal{O} j \]

- **constraints**
  
  \[ \Theta ::= \epsilon \mid c \cup \Theta \]

Only handles constraints of the form \( u \mathcal{O} j \) 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 $\text{max}(i, j)$ shouldn’t be fed back to it.
- To implement universe polymorphism, must hack directly inside the kernel. Done for inductive types for now.
Introduction

The current setup
- Definitions
- Issues

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

The past & the future
Constraint checking

universe context \( \Psi \ ::= \overrightarrow{i} \succeq \Theta \)

Constraints are generated once at \textit{refinement} time (outside the kernel):

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

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

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

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

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

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

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

\[
\text{INFER-CST} \quad \begin{align*}
(id : T) & \in \Sigma \\
\Gamma; \Psi & \vdash id \uparrow \rightsquigarrow \Psi \vdash id : T
\end{align*}
\]
Constraints are generated once at refinement time (outside the kernel):

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

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

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

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

- 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 \( \text{id} := t : T \).

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

\( \Rightarrow \) Guiding principle: constants are transparent, indistinguishable from their bodies.
To use $id$, we change elaboration of constants to:

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

$\Rightarrow$ Constants now carry their universe substitution/instance.

$\Rightarrow$ Inductives and constructors treated the same way.
Universe Polymorphic definitions: conversion

**R-δ-L**

\[
\frac{c \rightarrow i}{t} \quad \frac{t \rightarrow d}{\psi} \quad \frac{d \rightarrow R}{\psi} \quad u
\]

**R-δ-R**

\[
\frac{c \rightarrow i}{u} \quad \frac{t \rightarrow \psi}{R} \quad \frac{d \rightarrow \psi}{R} \quad u
\]

**R-FO**

\[
\frac{a \rightarrow b}{\psi} \quad \frac{u \rightarrow v}{\psi} \quad \frac{v \rightarrow v}{R \psi}
\]

\[
\frac{c \rightarrow u}{a \rightarrow b} \quad \frac{a \rightarrow c}{R \psi} \quad \frac{c \rightarrow v}{b \rightarrow \psi}
\]
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 \(\Rightarrow\) more functional.
- User-level control on generated universes and form of constraints (simplification, declaration...).
- Mixing polymorphic and monomorphic definitions.
Design choices

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 $id_i$ and $id_j$: 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** \( U_1 : U_2 := \text{Type}_j \leadsto j < i \)

**Definition** \( U_0 : U_1 := \text{Type}_k \leadsto k < j \)

**Definition** \( U_{02} : U_2 := U_0 \leadsto k < i \)

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

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 : ?rigid:rigid -> env -> evar_map ->
global_reference -> evar_map * constr

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

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

\[
\begin{align*}
\text{Elab-R-δ-left} & \\
\frac{c \downarrow i \rightarrow_{\delta} t \quad t \overset{R}{\rightarrow_{\psi}} u \leadsto \psi'}{c \downarrow i \rightarrow_{\delta} \overset{R}{\rightarrow_{\psi}} u \leadsto \psi'}
\end{align*}
\]

\[
\begin{align*}
\text{Elab-R-δ-right} & \\
\frac{c \downarrow i \rightarrow_{\delta} u \quad t \equiv^R_{\psi} u \overset{R}{\rightarrow_{\psi}} \leadsto \psi'}{t \equiv^R_{\psi} c \downarrow i \rightarrow_{\delta} \leadsto \psi'}
\end{align*}
\]

\[
\begin{align*}
\text{Elab-R-FO} & \\
\frac{\overset{R}{\rightarrow_{\psi}} a \, s \equiv \psi \overset{R}{\rightarrow_{\psi}} b s \leadsto \psi'}{\overset{R}{\rightarrow_{\psi}} a \, s \equiv \psi \overset{R}{\rightarrow_{\psi}} b s \leadsto \psi''}
\end{align*}
\]

\[
\begin{align*}
\psi \models i \equiv j \leadsto \psi' & : \text{unification of universe instances.}
\end{align*}
\]

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

\[
\begin{align*}
\text{Elab-Univ-Flexible} & \\
\frac{i_f \lor j_f \in \overset{\rightarrow}{u_s} \quad \psi \land i = j \models}{(\overset{\rightarrow}{u_s} \models \psi) \models i \equiv j \leadsto \psi \land i = j}
\end{align*}
\]
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

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

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

Levels only, adding constraint if an algebraic would appear:

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

That’s a lot of fresh universe variables!!

Typical example:

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

That’s a lot of fresh universe variables!!

Typical example:

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

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

Typical example:

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

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

\[\Rightarrow\] 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 \( i \) (\( j \cap i \)).
- 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.*
Let $\text{false}_i : \text{Type}_{i+1} \triangleq (\Pi A : \text{Type}_i, A : \text{Type}_{\text{max}(i+1,i)})$.

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

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

But $\text{false}_{\text{Prop}} \rightarrow^* \Pi A : \text{Prop}, A$, of type 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 `-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:

$$\text{subset}_i (A : \text{Type}_i) (P : A \to \text{Prop}) : \text{Type}_i \triangleq (\sum_{i, \text{Prop}} A P : \text{Type}_{\text{max}(i, \text{Prop})})$$
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: \( \text{nf_evars_universes} : \text{evar_map} \to \text{constr} \to \text{constr} \)
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
Harper and Pollack (TCS’91). Handling of definitions and typical ambiguity in type synthesis.

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 *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 *declare* universes and constraints (no user syntax yet).
- Resizing rules.
That’s all folks!
At the end of elaboration: \( \overrightarrow{i} \models \Theta \vdash t : T \).

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: \( \overrightarrow{i} \models \Theta \vdash t : T \).
Find a minimal set of universes variables \( \overrightarrow{i} \subseteq \overrightarrow{i}' \), 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.
- 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”. 

At the end of elaboration: \( i \vdash \Theta \vdash t : T \).

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

- 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
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: $\max(\vec{j})$, $j \not\in i \in \Theta$. If $i$ has no lower constraints it must be kept.
- Generate upper constraints $\{\text{glb } j \mid i \not\in j \in \Theta\}$
- Set $i := \text{glb}$ except if $\text{glb}$ algebraic and $i$ has upper constraints. We can share such $\text{glbs}$ though.