Universe Polymorphism & Fast Projections
Matthieu Sozeau

Coq Working Group
November 25th 2013
Paris
Without polymorphism

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

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

\[ \not\vdash \text{id} \ (\Pi(A : \text{Type}_{l}), \ A \rightarrow A) \ \text{id} : (\Pi(A : \text{Type}_{l}), \ A \rightarrow A) \]
Universe Polymorphism & Fast Projections

1. The current setup
   - Definitions
   - Issues

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

3. The past & the future

4. Fast projections, quickly
Implicit universes with cumulativity, à la Russell.

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

\[
\begin{align*}
\text{PROD} & \quad \Gamma; \Theta \vdash T : \text{Type}_i \rightsquigarrow \Theta_1 \quad \Gamma, x : T; \Theta_1 \vdash U : \text{Type}_j \rightsquigarrow \Theta_2 \\
& \quad \Gamma; \Theta \vdash \Pi x : T.U : \text{Type}_{\text{max}(i,j)} \rightsquigarrow \Theta_2 \\
\text{CONV} & \quad \Gamma; \Theta \vdash t : U \rightsquigarrow \Theta_1 \\
& \quad \Gamma; \Theta_1 \vdash V : s \rightsquigarrow \Theta_2 \\
& \quad \Theta_2 \vdash U \leq V \rightsquigarrow \Theta_3 \\
& \quad \Gamma; \Theta \vdash t : V \rightsquigarrow \Theta_3
\end{align*}
\]
**Kernel Conversion**

**Cumul-Sort**

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

**Cumul-Prod**

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

$$\Theta \vdash \Pi x : U.T \leq \Pi x : U'.T' \leadsto \Theta_2$$
Some definitions

Algebraic universes and constraints:

- **levels**: \( i, j, le, lt \) \( \in \mathbb{N} \cup \{\text{Prop},\text{Set}\} \).
- **universes**: \( u, v \) ::= \( i \mid \max(le, lt) \).
- **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**
  
  \[ u, v \ ::= i \mid \max(l e, l t) \]

- **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).
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. The current setup
   - Definitions
   - Issues

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

3. The past & the future

4. Fast projections, quickly
universe context  $\Psi ::= \overrightarrow{i} \models \Theta$

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

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

Checking:  $\Gamma; \Psi \vdash t \downarrow T \rightsquigarrow \Psi' \vdash t' : T$
Constraint checking

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

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

**Check-Type**

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

**Infer-Cst**

\[
\frac{(id : T) \in \Sigma}{\Gamma; \Psi \vdash id \uparrow \leadsto \Psi \vdash id : T}
\]
universe context \( \Psi ::= \overrightarrow{i} \vdash \Theta \)

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

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

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

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

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

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

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

\[
\frac{\text{Infer-Cst}}{(\text{id} : \forall i \models \theta, T) \in \Sigma \quad \overrightarrow{i} : i \notin \overrightarrow{u}}{\Gamma; \overrightarrow{u} \models \Theta \vdash \text{id} \uparrow \leadsto \overrightarrow{u}, \overrightarrow{i} \models \Theta \cup \theta[\overrightarrow{i} / \overrightarrow{i}] \vdash \text{id}_{\overrightarrow{i}} : T[\overrightarrow{i} / \overrightarrow{i}]} \]

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

\[
\begin{align*}
R-\delta-L : \quad & \frac{c \rightarrow \delta \ t}{c \rightarrow \delta t} \quad t \overset{\delta}{\rightarrow} \overset{R_\psi}{\rightarrow} u \\
& \frac{c \rightarrow \delta t}{c \rightarrow \delta t} \quad \overset{R_\psi}{\rightarrow} u \\
R-\delta-R : \quad & \frac{c \rightarrow \delta \ u}{c \rightarrow \delta u} \quad t \overset{R_\psi}{=} u \overset{\delta}{\rightarrow} \overset{R_\psi}{\rightarrow} \\
& \frac{t \overset{R_\psi}{=} c \rightarrow \delta u}{t \overset{R_\psi}{=} c \rightarrow \delta u} \\
R-FO : \quad & \frac{\overset{\rightarrow}{a} \overset{\rightarrow}{s} \overset{\rightarrow}{=}_\psi \overset{\rightarrow}{b} \overset{\rightarrow}{s}}{\frac{\overset{\rightarrow}{c} \overset{\rightarrow}{u} \overset{\rightarrow}{a} \overset{\rightarrow}{s} \overset{\rightarrow}{=} \overset{\rightarrow}{R}_\psi \overset{\rightarrow}{c} \overset{\rightarrow}{v} \overset{\rightarrow}{b} \overset{\rightarrow}{s}}{\overset{\rightarrow}{u} \overset{\rightarrow}{=} \overset{\rightarrow}{v}}}
\end{align*}
\]
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.
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: \( \text{id}_j U_02 \rightarrow^\ast (U_0 \rightarrow U_0) \) and \( \text{id}_i U_0 \rightarrow^\ast (U_0 \rightarrow U_0) \)

\[ \Rightarrow \text{Analyse variance of universes to relax first-order unification.} \]

\[ \text{Variance could help minimization too.} \]

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

\[ \Rightarrow \text{But it looks like the analysis is hard, not compositional and requires full normalisation of the constant bodies.} \]
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

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

**Elab-R-δ-left**

\[
\begin{align*}
& c \to \delta t \\
& t \overrightarrow{\alpha} \equiv_{\psi}^R u \leadsto \psi'
\end{align*}
\]

\[
\begin{align*}
& c \to \overrightarrow{\alpha} \equiv_{\psi}^R u \leadsto \psi'
\end{align*}
\]

**Elab-R-δ-right**

\[
\begin{align*}
& c \to \delta u \\
& t \equiv_{\psi}^R u \overrightarrow{\alpha} \leadsto \psi'
\end{align*}
\]

\[
\begin{align*}
& t \equiv_{\psi}^R c \to \overrightarrow{\alpha} \leadsto \psi'
\end{align*}
\]

**Elab-R-FO**

\[
\begin{align*}
& \overrightarrow{a}s \equiv_{\psi} bs \leadsto \psi' \\
& \psi' \models \overrightarrow{u} \equiv \overrightarrow{v} \leadsto \psi''
\end{align*}
\]

\[
\begin{align*}
& c \overrightarrow{u} \overrightarrow{a}s \equiv_{\psi}^R \overrightarrow{c} \overrightarrow{b}s \leadsto \psi'
\end{align*}
\]

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

**Elab-Univ-Eq**

\[
\begin{align*}
& \psi \models i = j \\
& \psi \models i \equiv j \leadsto \psi
\end{align*}
\]

**Elab-Univ-Flexible**

\[
\begin{align*}
& i_f \lor j_f \in \overrightarrow{u}_s \\
& \psi \land i = j \models
\end{align*}
\]

\[
\begin{align*}
& (\overrightarrow{u}_s \models \psi) \models i \equiv j \leadsto \psi \land i = j
\end{align*}
\]
Universe instances are levels: Suppose

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

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

Levels only, adding constraint if an algebraic would appear:

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

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

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

Levels only, adding constraint if an algebraic would appear:

\[\Gamma; \overrightarrow{u} \vdash \theta \vdash \text{id} (\Sigma A P) \uparrow \overrightarrow{u}, k \vdash \theta \cup \max(i, j) \leq k \vdash \text{id}_k (\Sigma_{i,j} A P) \ldots\]
That’s a lot of fresh universe variables!!

Typical example:

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

That’s a lot of fresh universe variables!!

Typical example:

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

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

That’s a lot of fresh universe variables!!

Typical example:

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

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

$$\Rightarrow$$ 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 $$i$$ ($$j \aleph i$$).
- 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 \textit{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 \text{Prop} by impredicativity (and \text{Type}_{\text{Prop}+1} still).
Dealing with Prop

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

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).
Partial Solutions:

- Allow to disable \texttt{Prop} completely, with \texttt{-no-prop}, for HoTTists.
- Disallow instantiating a parameter level $i$ with \texttt{Prop}, similar to the current \texttt{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:

\[
\text{subset}_i \ (A : \text{Type}_i) \ (P : A \rightarrow \text{Prop}) : \text{Type}_i \triangleq \\
(\Sigma_{i,\text{Prop}} A \ P : \text{Type}_{\max(i,\text{Prop})})
\]
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:

\[
\text{nf_evars_universes : evar_map} \rightarrow \text{constr} \rightarrow \text{constr}
\]
Universe polymorphism in Coq

1. The current setup
   - Definitions
   - Issues

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

3. The past & the future

4. Fast projections, quickly
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.
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.
Fast projections

1. The current setup
   - Definitions
   - Issues

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

3. The past & the future

4. Fast projections, quickly
Motivated by Agda’s experience, Garillot’s PhD. Set Primitive Projections can be used to declare projections as primitive instead of as regular case-analysis constants. The two can be mixed.

```
type projection = kernel_name (* * int *)

type constr = ...
  | Proj of projection * constr

⇒ Always applied to their record object, no parameters, no universe instance.
```
with huge performance implications

**Fast typechecking** (kinda bidirectional):

\[
\frac{\text{Proj} \quad \Gamma \vdash c : I \ p \vdash s}{\Gamma \vdash c.(p) \triangleq \text{Proj}(p, c) : (1.p)[p \vdash s]}
\]

**Fast reduction** (could be faster):

\[
\text{Proj}(p, \text{mkI} \ \overline{\text{par} \vdash s \ \text{args} \vdash s}) \rightarrow_{l} \text{args}.(i) \quad \text{where } i = \Gamma(p).\text{arg}
\]

Never see a dummy case analysis unfolded again!
Conversion, \( \eta \)-rule

Adds \( \eta \) for records by giving an easily recognizable canonical form:

\[
c = \text{mkI} \xrightarrow{\text{pars}} c.(p_1) \ldots c.(p_n) \quad \text{when I is a non-empty record}
\]

The trick is the same as for \( \lambda \): when converting a rigid constructor (\text{mkI}) with a variable, expand the variable to the constructor application (using the \( \text{pars} \) information that is available).
Now projections always have to be applied. What if you used @p before?

- @p becomes a constant name, the eta-expansion of the projection viewed as a constant, it reduces to the projection. It simplifies if the record argument is a constructor, even if the parameter arguments get lost. This provides almost complete source compatibility.

- r.(p) (or r.p if we can agree) is record projection p applied to r, whatever the implicit status of r for p is. The remaining arguments are parsed according to the implicits declared for the projection.

- r.(@p ...) = @p ... r but @p ... r is no longer syntactically equal to r.(p).

Incompatibilities: mainly due to Notations using p applied to explicit parameters ⇒ use @p.
Exponential speedup on typechecking, conversion etc... for nested records and dependent sums.

- Our groupoid implementation went from 20 minutes to 6 seconds when we changed representation.
- Small benefits in the stdlib (field/ring etc based on records).
- Probably huge in categorical formalizations, libraries based on HoTT and anything that uses sigma types or parameterized records heavily.

TODO: properly adapt \texttt{vm\_compute} to projections and universes, only full normalization to a canonical form made of constructors is correct for now. Native compilation adapted by M. Dénès.
That’s all folks!
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 \( \sigma : \vec{i} \to \vec{u} \) 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: \( \overrightarrow{i} \models \Theta \vdash t : T \).
Find a minimal set of universes variables \( \overrightarrow{i}' \subset \overrightarrow{i} \), universes \( \overrightarrow{u} \), a substitution \( \sigma : \overrightarrow{i} \to \overrightarrow{u} \) 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: \( \vec{i} \models \Theta \vdash t : T \).

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: $\max(\vec{j}), j \cup i \in \Theta$. If $i$ has no lower constraints it must be kept.
- Generate upper constraints $\{\text{glb} \cup j \mid i \cup j \in \Theta\}$
- Set $i := \text{glb}$ except if $\text{glb}$ algebraic and $i$ has upper constraints. We can share such glbs though.