The Predicative, Polymorphic, Cumulative Calculus of Inductive Constructions and its implementation

Matthieu Sozeau

Inria Paris & IRIF, France

June 20th, 2018

TYPES 2018, Braga, Portugal

What are universes?

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

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

How are they organised?

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

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

Rules of CIC

Typing of universes, formally (Russell-style):

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

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

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: assume the reader could infer levels for universes and check consistency. Easier said than done!
- On computer: let the computer infer levels and check consistency in the background.

Floating universes

Formally, translate from anonymous Types to explicit $\mathtt{Type}_i$s.
But in general many i's can work!

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

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

or

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

or . . . ?

$$\Rightarrow \text{universe variables}$$

Floating universes and constraints

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

$$\textsc{Type-intro}\quad \frac{\vdash_{\mathit{float}} \Gamma \quad (i, j \in \mathbb{L})}{\Gamma \vdash_{\mathit{float}} \mathtt{Type}_i : \mathtt{Type}_j \rightsquigarrow i < j}$$

$$\textsc{Prod-intro}\quad \frac{\Gamma \vdash_{\mathit{float}} A : \mathtt{Type}_i \qquad \Gamma, x : A \vdash B : \mathtt{Type}_j}{\Gamma \vdash_{\mathit{float}} \Pi x : A.B : \mathtt{Type}_k \rightsquigarrow \max(i, j) \leq k}$$

Floating universes and constraints II

Correctness

If $\Gamma \vdash_{float} t : T \rightsquigarrow \Theta$ and Θ is satisfiable (with assignemnt σ, i.e. $\models \Theta[\sigma]$) then $\Gamma[\sigma] \vdash t[\sigma] : T[\sigma]$.

Constraints $\Rightarrow$ graph structure with $i \leq j$ or $i < j$ edges
Consistency $\Rightarrow$ strict cycle freeness

Implementation note: checked incrementaly using an algorithm of Tarjan et al. Verification in progress by A. Guéneau and J.H. Jourdan.

Without polymorphism

Floating levels give a false sense of polymorphism:

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

$$\rightsquigarrow\ \vdash_{\mathit{float}} \mathrm{id} : \Pi(A : \texttt{Type}_l),\ A \to A : \texttt{Type}_{l+1}$$

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

$$\forall_{\mathit{float}}\ \mathrm{id}\ (\Pi(A : \texttt{Type}_l),\ A \to A)\ \mathrm{id} : \tau$$

Because $l + 1 \not\leq l$.

With polymorphism

Real, bounded polymorphism:

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

or equivalently:

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

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

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

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

Universes in Coq

Elaboration

Harper & Pollack, Type Checking with Universes, TCS'91

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.

$$\text{universe context} \quad \Psi \quad ::= \quad \vec{i} \models \Theta$$

Elaboration

Harper & Pollack, Type Checking with Universes, TCS'91

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.

$$\text{universe context} \quad \Psi \quad ::= \quad \vec{i} \models \Theta$$

Elaboration in bidirectional fashion:

- Inference: $\Gamma; \Psi \vdash t \Uparrow \rightsquigarrow \Psi' \vdash t' : T$
- Checking: $\Gamma; \Psi \vdash t \Downarrow T \rightsquigarrow \Psi' \vdash t' : T$

Elaboration

Harper & Pollack, Type Checking with Universes, TCS'91

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.

$$\text{universe context} \quad \Psi \quad ::= \quad \vec{i} \vDash \Theta$$

Elaboration in bidirectional fashion:

- Inference: $\Gamma; \Psi \vdash t \Uparrow \rightsquigarrow \Psi' \vdash t' : T$
- Checking: $\Gamma; \Psi \vdash t \Downarrow T \rightsquigarrow \Psi' \vdash t' : T$

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

Introducing universe polymorphic definitions

Sozeau & Tabareau, Universe Polymorphism in Coq, ITP'14

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

Introducing universe polymorphic definitions

Sozeau & Tabareau, Universe Polymorphism in Coq, ITP'14

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

1. $\Gamma; \vdash T \Uparrow \rightsquigarrow \Psi \vdash T' : s$

Introducing universe polymorphic definitions

Sozeau & Tabareau, Universe Polymorphism in Coq, ITP'14

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

1. $\Gamma; \vdash T \Uparrow \rightsquigarrow \Psi \vdash T' : s$
2. $\Gamma; \Psi \vdash t \Downarrow T' \rightsquigarrow i \vDash \theta \vdash t : T'$

Introducing universe polymorphic definitions

Sozeau & Tabareau, Universe Polymorphism in Coq, ITP'14

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

1. $\Gamma; \vdash T \Uparrow \rightsquigarrow \Psi \vdash T' : s$
2. $\Gamma; \Psi \vdash t \Downarrow T' \rightsquigarrow i \vDash \theta \vdash t : T'$
3. Add $\mathrm{id} : \forall\ i \vDash \theta, T' := t$ to the environment.

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

Using universe polymorphic definitions

$$\textsc{Infer-Cst}\quad \frac{}{\Gamma; \vec{u} \vDash \Theta \vdash \mathrm{id} \Uparrow \rightsquigarrow \Theta \cup \theta[\vec{I}/\vec{i}] \vdash \mathrm{id}_{\vec{I}} : T[\vec{I}/\vec{i}]}$$

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

$\Rightarrow$ Inductives and constructors treated the same way.

Conversion and cumulativity

$$R \in \{=, \leq\}$$

$$\textsc{Cumul-Sort}\ \frac{\psi \vDash i\ R\ j}{\mathtt{Type}_i =^R_\psi \mathtt{Type}_j}$$

Conversion and cumulativity

$$R \in \{=, \leq\}$$

$$\textsc{Cumul-Sort}\ \frac{\psi \vDash i\ R\ j}{\mathtt{Type}_i =^R_\psi \mathtt{Type}_j}$$

$$\textsc{Cumul-Prod}\ \frac{U =^=_\psi U' \qquad T =^R_\psi T'}{\Pi x : U.T =^R_\psi \Pi x : U'.T'}$$

Conversion and cumulativity

$$R \in \{=, \leq\}$$

$$\textsc{Cumul-Sort}\ \frac{\psi \vDash i\ R\ j}{\mathtt{Type}_i =^R_\psi \mathtt{Type}_j} \qquad \textsc{Cumul-Prod}\ \frac{U =^=_\psi U' \qquad T =^R_\psi T'}{\Pi x : U.T =^R_\psi \Pi x : U'.T'}$$

$$\textsc{Conv-FO}\ \frac{\vec{a} =^=_\psi \vec{b} \qquad \psi \models \vec{u} = \vec{v}}{\mathtt{c}_{\vec{u}}\ \vec{a} =^R_\psi \mathtt{c}_{\vec{v}}\ \vec{b}}$$

Uses backtracking. Essential rule to avoid going to normal forms: definitions treated as abstractions.

Unification and cumulativity

$$\texttt{Definition}\ U2 := \texttt{Type}_i.$$
$$\texttt{Definition}\ U1 : U2 := \texttt{Type}_j \rightsquigarrow j < i$$

Unification and cumulativity

$$\texttt{Definition}\ U2 := \texttt{Type}_i.$$
$$\texttt{Definition}\ U1 : U2 := \texttt{Type}_j \rightsquigarrow j < i$$
$$\texttt{Definition}\ U0 : U1 := \texttt{Type}_k \rightsquigarrow k < j$$

Unification and cumulativity

$$\texttt{Definition}\ U2 := \texttt{Type}_i.$$
$$\texttt{Definition}\ U1 : U2 := \texttt{Type}_j \rightsquigarrow j < i$$
$$\texttt{Definition}\ U0 : U1 := \texttt{Type}_k \rightsquigarrow k < j$$
$$\texttt{Definition}\ U02 : U2 := U0 \rightsquigarrow k < i$$

Unification and cumulativity

$\texttt{Definition}\ U2 := \texttt{Type}_i.$
$\texttt{Definition}\ U1 : U2 := \texttt{Type}_j \rightsquigarrow j < i$
$\texttt{Definition}\ U0 : U1 := \texttt{Type}_k \rightsquigarrow k < j$
$\texttt{Definition}\ U02 : U2 := U0 \rightsquigarrow k < i$

Unification of id_i and id_j:

$$\mathrm{id}_j\ U0 \equiv^{=} \mathrm{id}_i\ U02 \rightsquigarrow i = j$$

Unification and cumulativity

$\texttt{Definition}\ U2 := \texttt{Type}_i.$
$\texttt{Definition}\ U1 : U2 := \texttt{Type}_j \rightsquigarrow j < i$
$\texttt{Definition}\ U0 : U1 := \texttt{Type}_k \rightsquigarrow k < j$
$\texttt{Definition}\ U02 : U2 := U0 \rightsquigarrow k < i$

Unification of id_i and id_j:

$$\mathrm{id}_j\ U0 \equiv^{=} \mathrm{id}_i\ U02 \rightsquigarrow i = j$$

But:

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

No constraint is actually necessary if we reduce to normal form!

Least-commitment principle

Use two kinds of universe level variables during elaboration:

- Polymorphic constants get elaborated with fresh flexible argument levels.
- Typical ambiguity (e.g. `Type`) creates rigid variables.
- User-given levels are rigid

Unification with universes

Ziliani & Sozeau, ICFP'15

$t \equiv^{R}_{\psi} u \rightsquigarrow \psi'$: unification of t and u under ψ.

$$\frac{\vec{a} \equiv^{=}_{\psi} \vec{b} \rightsquigarrow \psi' \qquad \psi' \models \vec{u} \equiv \vec{v} \rightsquigarrow \psi''}{\mathrm{c}_{\vec{u}}\ \vec{a} \equiv^{\mathbf{R}}_{\psi} \mathrm{c}_{\vec{v}}\ \vec{b} \rightsquigarrow \psi'}$$

ELAB-R-FO

Unification with universes

Ziliani & Sozeau, ICFP'15

$t \equiv^{R}_{\psi} u \rightsquigarrow \psi'$: unification of t and u under ψ.

$$\frac{\vec{a} \equiv^{=}_{\psi} \vec{b} \rightsquigarrow \psi' \qquad \psi' \models \vec{u} \equiv \vec{v} \rightsquigarrow \psi''}{\underline{c}_{\vec{u}}\ \vec{a} \equiv^{\mathbf{R}}_{\psi} \underline{c}_{\vec{v}}\ \vec{b} \rightsquigarrow \psi'}$$

Elab-R-FO

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

$$\frac{\psi \models i = j}{\psi \models i \equiv j \rightsquigarrow \psi}$$

Elab-Univ-Eq

$$\frac{i_{\mathsf{f}} \vee j_{\mathsf{f}} \in \vec{u_s} \qquad \psi \wedge i = j \models}{(\vec{u_s} \models \psi) \models i \equiv j \rightsquigarrow \psi \wedge i = j}$$

Elab-Univ-Flexible

Minimization

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

Typical example:

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

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.

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.

$\Rightarrow$ Minimization: reduce the constraints to a minimal set of universe variables.

See Cardelli's greedy algorithm for $F^{\leq}$ inference, local type inference (Pierce & Turner, TOPLAS'00).

- Only applies to flexible variables.

Minimization, results

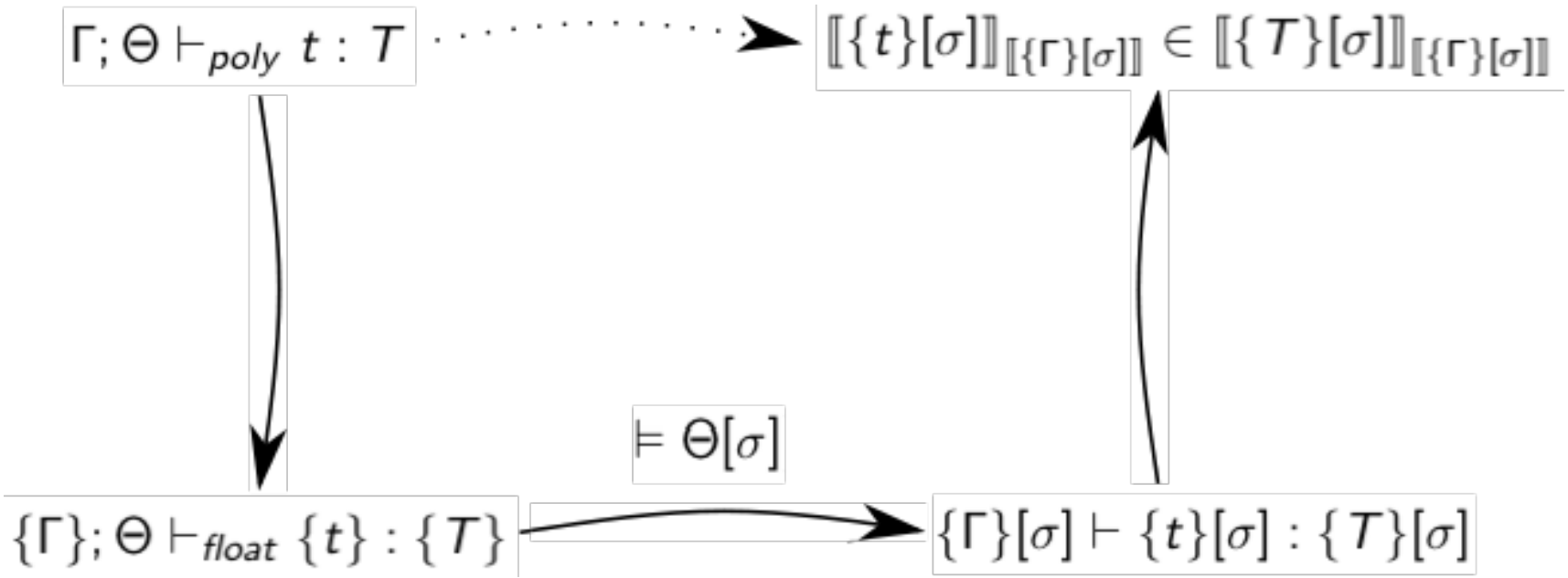
Correctness proof: easy, preservation of local solutions.

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

Theorem (*Correctness*)

If $\Gamma; \Theta \vdash_{poly} t : T$ *then* $\{\Gamma\}; \Theta \vdash_{float} \{t\} : \{T\}$ *where* $\{t\}$ *unfolds polymorphic definitions and makes copies of inductives.*

Summary


$\Gamma; \Theta \vdash_{poly} t : T$
$[\![\{t\}[\sigma]]\!]_{[\![\{\Gamma\}[\sigma]]\!]} \in [\![\{T\}[\sigma]]\!]_{[\![\{\Gamma\}[\sigma]]\!]}$
$\models \Theta[\sigma]$
$\{\Gamma\}; \Theta \vdash_{float} \{t\} : \{T\}$
$\{\Gamma\}[\sigma] \vdash \{t\}[\sigma] : \{T\}[\sigma]$

Cumulative Inductive Types

Universe polymorphism for inductive types

- Definitions can now be polymorphic in universe levels, e.g., categories:

```
Record Category@{i j} : Type@{max(i+1, j+1)} :=
  { Obj : Type@{i};
    Hom : Obj → Obj → Type@{j}; ... }.
```

Universe polymorphism for inductive types

- Definitions can now be polymorphic in universe levels, e.g., categories:

```
Record Category@{i j} : Type@{max(i+1, j+1)} :=
  { Obj : Type@{i};
    Hom : Obj → Obj → Type@{j}; ... }.
```

- Each universe polymorphic definition comes with constraints, e.g., for the category of categories:

```
Definition Cat@{i j k l} :=
  {| Obj := Category@{k l};
     Hom := fun C D ⇒ Functor@{k l k l} C D; ... |}
  : Category@{i j}.
```

with constraints:

$$k < i \quad \text{and} \quad l < i$$

Universe polymorphism on inductives

- Cumulativity is just conversion for inductives, i.e.:
  $\texttt{Category@\{i j\}} \preceq \texttt{Category@\{k l\}}$ iff $\texttt{i} = \texttt{k}$ and $\texttt{j} = \texttt{l}$

Universe polymorphism on inductives

- Cumulativity is just conversion for inductives, i.e.: $\texttt{Category@\{i j\}} \preceq \texttt{Category@\{k l\}}$ iff $\texttt{i} = \texttt{k}$ and $\texttt{j} = \texttt{l}$
- This means $\texttt{Cat@\{i j k l\}}$ is the category of all categories at $\{\texttt{k l}\}$ and *not lower* [1]
- Constraints on statements about universe polymorphic inductive definitions restrict to which copies they apply.

[1] There are however categories isomorphic to the categories in lower levels.

Relative size constraints

- For the category of categories `Cat@{i j k l}` the fact that it has exponentials has constraints $\mathtt{j} = \mathtt{k} = \mathtt{l}$:

$$\begin{gathered}\text{universe of objects } \mathtt{k} \\ = \\ \text{universe of morphisms } \mathtt{l} \\ = \\ \text{universe of functors between } \mathtt{k}\ \mathtt{l} \text{ categories}\end{gathered}$$

Relative size constraints

- For the category of categories `Cat@{i j k l}` the fact that it has exponentials has constraints `j` = `k` = `l`:

$$\begin{array}{c}\text{universe of objects } \texttt{k}\\ =\\ \text{universe of morphisms } \texttt{l}\\ =\\ \text{universe of functors between } \texttt{k}\ \texttt{l} \text{ categories}\end{array}$$

- In particular:

```
Definition Type_Cat@{i j} :=
  {| Obj := Type@{j};
     Hom := fun A B ⇒ A → B; ... |} : Category@{i j}.
```

with constraints $j < i$ is not an object of any copy of `Cat` with exponentials!

Relative size constraints

- For the category of categories `Cat@{i j k l}` the fact that it has exponentials has constraints $\texttt{j} = \texttt{k} = \texttt{l}$:

$$\begin{gathered}\text{universe of objects } \texttt{k} \\ = \\ \text{universe of morphisms } \texttt{l} \\ = \\ \text{universe of functors between } \texttt{k}\ \texttt{l} \text{ categories}\end{gathered}$$

- In particular:

```
Definition Type_Cat@{i j} :=
  {| Obj := Type@{j};
     Hom := fun A B ⇒ A → B; ... |} : Category@{i j}.
```

with constraints $j < i$ is not an object of any copy of `Cat` with exponentials!

- Yoneda embedding can't be simply defined as the exponential transpose of the *hom* functor

Inductive types in pCIC

$$\textsc{Ind}\ \frac{A \in Ar(s) \qquad \Gamma \vdash A : s' \qquad \Gamma, X : A \vdash C_i : s \qquad C_i \in Co(X)}{\Gamma \vdash \mathsf{Ind}(X : A)\{C_1, \ldots, C_n\} : A}$$

$Ar(s)$ is the set of types of the form: $\Pi \vec{x} : \vec{M}.\ s$

$Co(X)$ is the set of types of the form: $\Pi \vec{x} : \vec{M}.\ X\ \vec{m}$

Inductive types in pCIC

$$\textsc{Ind}\ \frac{A \in Ar(s) \qquad \Gamma \vdash A : s' \qquad \Gamma, X : A \vdash C_i : s \qquad C_i \in Co(X)}{\Gamma \vdash \mathsf{Ind}(X : A)\{C_1, \ldots, C_n\} : A}$$

$Ar(s)$ is the set of types of the form: $\Pi \vec{x} : \vec{M}.\ s$

$Co(X)$ is the set of types of the form: $\Pi \vec{x} : \vec{M}.\ X\ \vec{m}$

No Parameters (`T` in `vec T n`) are considered in this rule.

```
Inductive vec (T : Type) : nat → Type := nil : vec T 0
  | cons : forall n, T → vec T n → vec T (S n).
```

Inductive types in pCIC

$$\textsc{Ind}\ \frac{A \in Ar(s) \qquad \Gamma \vdash A : s' \qquad \Gamma, X : A \vdash C_i : s \qquad C_i \in Co(X)}{\Gamma \vdash \mathsf{Ind}(X : A)\{C_1, \ldots, C_n\} : A}$$

$Ar(s)$ is the set of types of the form: $\Pi \vec{x} : \vec{M}.\ s$

$Co(X)$ is the set of types of the form: $\Pi \vec{x} : \vec{M}.\ X\ \vec{m}$

No Parameters (`T` in `vec T n`) are considered in this rule.

```
Inductive vec (T : Type) : nat → Type := nil : vec T 0
  | cons : forall n, T → vec T n → vec T (S n).
```

Calculus of Cumulative Inductive Types (pCuIC)

Timany & Sozeau, FSCD'18

$$\textsc{C-Ind}\quad \frac{\begin{array}{c} I \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N}.\ s)\{\Pi\vec{x_1} : \vec{M_1}.\ X\ \vec{m_1}, \ldots, \Pi\vec{x_n} : \vec{M_n}.\ X\ \vec{m_n}\}) \\ I' \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N'}.\ s')\{\Pi\vec{x_1} : \vec{M'_1}.\ X\ \vec{m'_1}, \ldots, \Pi\vec{x_n} : \vec{M'_n}.\ X\ \vec{m'_n}\}) \\ \forall i.\ N_i \preceq N'_i \qquad \forall i, j.\ (M_i)_j \preceq (M'_i)_j \\ \mathit{length}(\vec{m}) = \mathit{length}(\vec{x}) \qquad \forall i.\ X\ \vec{m_i} \simeq X\ \vec{m'_i} \end{array}}{I\ \vec{m} \preceq I'\ \vec{m}}$$

Calculus of Cumulative Inductive Types (pCuIC)

Timany & Sozeau, FSCD'18

$$\textsc{C-Ind}\quad \frac{\begin{array}{c} I \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N}.\ s)\{\Pi\vec{x_1} : \vec{M_1}.\ X\ \vec{m_1}, \ldots, \Pi\vec{x_n} : \vec{M_n}.\ X\ \vec{m_n}\}) \\ I' \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N'}.\ s')\{\Pi\vec{x_1} : \vec{M'_1}.\ X\ \vec{m'_1}, \ldots, \Pi\vec{x_n} : \vec{M'_n}.\ X\ \vec{m'_n}\}) \\ \forall i.\ N_i \preceq N'_i \qquad \forall i, j.\ (M_i)_j \preceq (M'_i)_j \\ \mathit{length}(\vec{m}) = \mathit{length}(\vec{x}) \qquad \forall i.\ X\ \vec{m_i} \simeq X\ \vec{m'_i} \end{array}}{I\ \vec{m} \preceq I'\ \vec{m}}$$

Calculus of Cumulative Inductive Types (pCuIC)

Timany & Sozeau, FSCD'18

$$\textsc{C-Ind}\quad \frac{\begin{array}{c} I \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N}.\ s)\{\Pi\vec{x_1} : \vec{M_1}.\ X\ \vec{m_1}, \ldots, \Pi\vec{x_n} : \vec{M_n}.\ X\ \vec{m_n}\}) \\ I' \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N'}.\ s')\{\Pi\vec{x_1} : \vec{M'_1}.\ X\ \vec{m'_1}, \ldots, \Pi\vec{x_n} : \vec{M'_n}.\ X\ \vec{m'_n}\}) \\ \forall i.\ N_i \preceq N'_i \qquad \forall i, j.\ (M_i)_j \preceq (M'_i)_j \\ \mathit{length}(\vec{m}) = \mathit{length}(\vec{x}) \qquad \forall i.\ X\ \vec{m_i} \simeq X\ \vec{m'_i} \end{array}}{I\ \vec{m} \preceq I'\ \vec{m}}$$

Calculus of Cumulative Inductive Types (pCuIC)

Timany & Sozeau, FSCD'18

$$\textsc{C-Ind}\quad \frac{\begin{array}{c} I \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N}.\ s)\{\Pi\vec{x_1} : \vec{M_1}.\ X\ \vec{m_1}, \ldots, \Pi\vec{x_n} : \vec{M_n}.\ X\ \vec{m_n}\}) \\ I' \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N'}.\ s')\{\Pi\vec{x_1} : \vec{M'_1}.\ X\ \vec{m'_1}, \ldots, \Pi\vec{x_n} : \vec{M'_n}.\ X\ \vec{m'_n}\}) \\ \forall i.\ N_i \preceq N'_i \qquad \forall i, j.\ (M_i)_j \preceq (M'_i)_j \\ \mathit{length}(\vec{m}) = \mathit{length}(\vec{x}) \qquad \boxed{\forall i.\ X\ \vec{m_i} \simeq X\ \vec{m'_i}} \end{array}}{I\ \vec{m} \preceq I'\ \vec{m}}$$

Calculus of Cumulative Inductive Types (pCuIC)

Timany & Sozeau, FSCD'18

$$\textsc{C-Ind}\quad \frac{\begin{array}{c} I \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N}.\ s)\{\Pi\vec{x_1} : \vec{M_1}.\ X\ \vec{m_1}, \ldots, \Pi\vec{x_n} : \vec{M_n}.\ X\ \vec{m_n}\}) \\ I' \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N'}.\ s')\{\Pi\vec{x_1} : \vec{M'_1}.\ X\ \vec{m'_1}, \ldots, \Pi\vec{x_n} : \vec{M'_n}.\ X\ \vec{m'_n}\}) \\ \forall i.\ N_i \preceq N'_i \qquad \forall i, j.\ (M_i)_j \preceq (M'_i)_j \\ \mathit{length}(\vec{m}) = \mathit{length}(\vec{x}) \qquad \forall i.\ X\ \vec{m_i} \simeq X\ \vec{m'_i} \end{array}}{I\ \vec{m} \preceq I'\ \vec{m}}$$

Calculus of Cumulative Inductive Types (pCuIC)

Timany & Sozeau, FSCD'18

$$\textsc{C-Ind}\quad \frac{\begin{array}{c} I \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N}.\ s)\{\Pi\vec{x_1} : \vec{M_1}.\ X\ \vec{m_1}, \ldots, \Pi\vec{x_n} : \vec{M_n}.\ X\ \vec{m_n}\}) \\ I' \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N'}.\ s')\{\Pi\vec{x_1} : \vec{M'_1}.\ X\ \vec{m'_1}, \ldots, \Pi\vec{x_n} : \vec{M'_n}.\ X\ \vec{m'_n}\}) \\ \forall i.\ N_i \preceq N'_i \qquad \forall i, j.\ (M_i)_j \preceq (M'_i)_j \\ \mathit{length}(\vec{m}) = \mathit{length}(\vec{x}) \qquad \forall i.\ X\ \vec{m_i} \simeq X\ \vec{m'_i} \end{array}}{I\ \vec{m} \preceq I'\ \vec{m}}$$

Calculus of Cumulative Inductive Types (pCuIC)

Timany & Sozeau, FSCD'18

$$\textsc{C-Ind}\quad \frac{\begin{array}{c} I \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N}.\ s)\{\Pi\vec{x_1} : \vec{M_1}.\ X\ \vec{m_1}, \ldots, \Pi\vec{x_n} : \vec{M_n}.\ X\ \vec{m_n}\}) \\ I' \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N'}.\ s')\{\Pi\vec{x_1} : \vec{M'_1}.\ X\ \vec{m'_1}, \ldots, \Pi\vec{x_n} : \vec{M'_n}.\ X\ \vec{m'_n}\}) \\ \forall i.\ N_i \preceq N'_i \qquad \forall i, j.\ (M_i)_j \preceq (M'_i)_j \\ \mathit{length}(\vec{m}) = \mathit{length}(\vec{x}) \qquad \forall i.\ X\ \vec{m_i} \simeq X\ \vec{m'_i} \end{array}}{I\ \vec{m} \preceq I'\ \vec{m}}$$

- Example:

$$\texttt{Category@\{i j\}} \equiv \mathsf{Ind}(X : \texttt{Type}_{max(i+1,j+1)})\ \{\Pi o : \texttt{Type}_i.\Pi h : o \to o \to \texttt{Type}_j.\cdots\}$$

- By C-Ind:

$$\texttt{i} \leq \texttt{k} \text{ and } \texttt{j} \leq \texttt{l} \Rightarrow \texttt{Category@\{i j\}} \preceq \texttt{Category@\{k l\}}$$

Calculus of Cumulative Inductive Types (pCuIC)

Timany & Sozeau, FSCD'18

$$\textsc{C-Ind}\quad \frac{\begin{array}{c} I \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N}.\ s)\{\Pi\vec{x_1} : \vec{M_1}.\ X\ \vec{m_1}, \ldots, \Pi\vec{x_n} : \vec{M_n}.\ X\ \vec{m_n}\}) \\ I' \equiv (\mathsf{Ind}(X : \Pi\vec{x} : \vec{N'}.\ s')\{\Pi\vec{x_1} : \vec{M'_1}.\ X\ \vec{m'_1}, \ldots, \Pi\vec{x_n} : \vec{M'_n}.\ X\ \vec{m'_n}\}) \\ \forall i.\ N_i \preceq N'_i \qquad \forall i, j.\ (M_i)_j \preceq (M'_i)_j \\ \mathit{length}(\vec{m}) = \mathit{length}(\vec{x}) \qquad \forall i.\ X\ \vec{m_i} \simeq X\ \vec{m'_i} \end{array}}{I\ \vec{m} \preceq I'\ \vec{m}}$$

- Example:

$$\texttt{Category@\{i j\}} \equiv \mathsf{Ind}(X : \texttt{Type}_{max(i+1,j+1)})\ \{\Pi o : \texttt{Type}_i.\Pi h : o \to o \to \texttt{Type}_j.\cdots\}$$

- By C-Ind:

$$\texttt{i} \leq \texttt{k} \text{ and } \texttt{j} \leq \texttt{l} \Rightarrow \texttt{Category@\{i j\}} \preceq \texttt{Category@\{k l\}}$$

- C-Ind does not consider the parameters or sorts of I and I'

Back to categories

- For `Cat@{i j k l}` with exponentials we had the constraints: $\mathtt{j} = \mathtt{k} = \mathtt{l}$
- `Type_Cat@{i' j'}` we had the constraint: $j' < i'$
- Now `Type_Cat@{i' j'} : Obj Cat@{i j k l}` just imposes the constraint: $\mathtt{i}' \leq \mathtt{k} \wedge \mathtt{j}' \leq \mathtt{l}$, which is consistent.

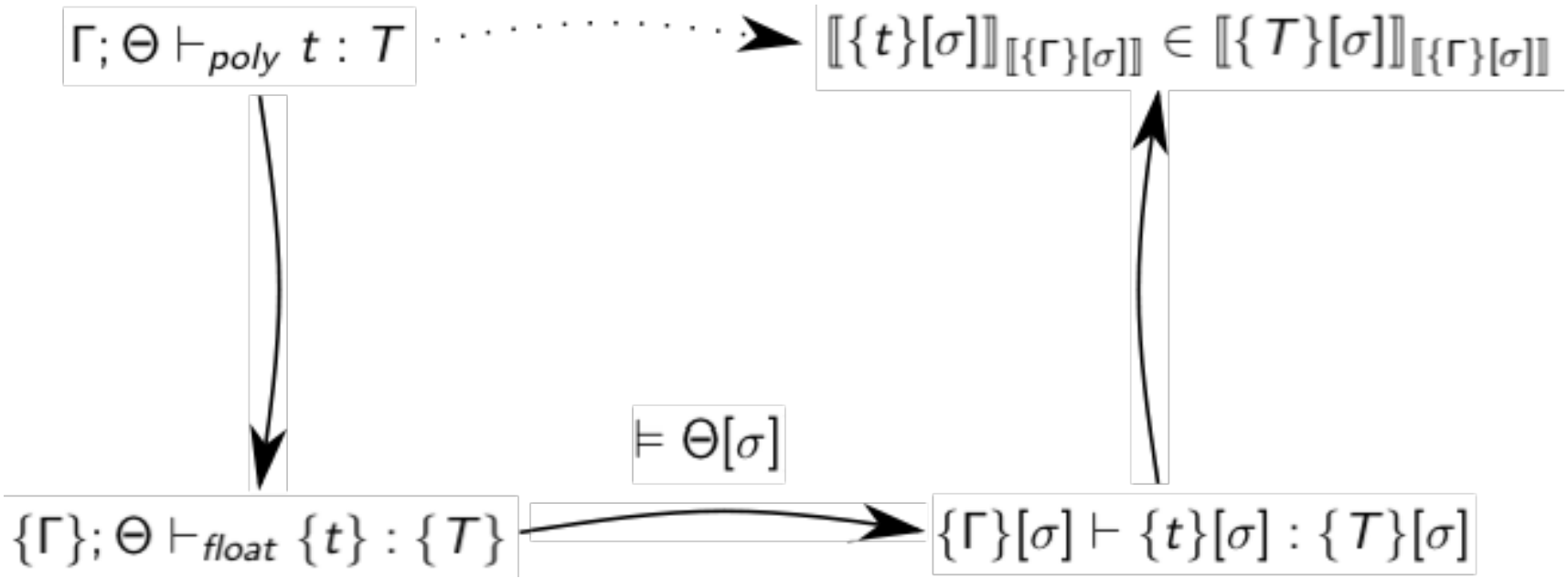
Theoretical justification

We construct a set theoretic model for pCuIC
$[\![\cdot]\!] : \mathit{Terms}_{\mathit{pCuIC}} \to \mathit{ZFC}$: ZFC with suitable axioms, e.g., inaccessible cardinals, to model pCuIC universes.

- Based on a modification of Werner & Lee's model which assumed SN, which we avoid.
- Supports `Prop` $\preceq$ `Type` through trace encoding (Aczel's trick).
- For subtyping $A \preceq B$ we have $[\![A]\!] \subseteq [\![B]\!]$
- Inductive types interpreted using least fixpoints of **monotone**[2] functions, presented with eliminators instead of match+fix+guard condition.
- This justifies C-Ind and a relaxed rule on constructors.

[2] Due to strict positivity condition

Back-and-forth with the model



Conversion from cumulativity

- In pCuIC we consider *fully applied* inductive types $I\ \vec{m}$ and $I'\ \vec{m}$ convertible if they are mutually subtypes

$$\textsc{Conv-Ind}\ \frac{I\ \vec{m} \preceq I'\ \vec{m} \qquad I'\ \vec{m} \preceq I\ \vec{m}}{I\ \vec{m} \simeq I'\ \vec{m}}$$

Conversion from cumulativity

- In pCuIC we consider *fully applied* inductive types $I\ \vec{m}$ and $I'\ \vec{m}$ convertible if they are mutually subtypes

$$\textsc{Conv-Ind}\ \frac{I\ \vec{m} \preceq I'\ \vec{m} \qquad I'\ \vec{m} \preceq I\ \vec{m}}{I\ \vec{m} \simeq I'\ \vec{m}}$$

- $\texttt{i} = \texttt{k}$ and $\texttt{j} = \texttt{l} \Rightarrow \texttt{Category@\{i j\}} \simeq \texttt{Category@\{k l\}}$
- For *constructors*: convertible as long as they are compared at the *same supertype*.

Data structures and "template polymorphism"

$$\texttt{list@\{i\}}\ (\texttt{A} : \texttt{Type}_i) \equiv \mathsf{Ind}(X : \texttt{Type}_i)\{X, A \to X \to X\}$$

- By C-Ind:

  $$\texttt{list@\{i\}}\ \texttt{A} \preceq \texttt{list@\{j\}}\ \texttt{A}$$

Data structures and "template polymorphism"

$$\texttt{list@\{i\}}\ (\texttt{A} : \texttt{Type}_i) \equiv \mathsf{Ind}(X : \texttt{Type}_i)\{X, A \to X \to X\}$$

- By C-Ind:

  $\texttt{list@\{i\}}\ \texttt{A} \preceq \texttt{list@\{j\}}\ \texttt{A}$ (**regardless of** `i` **and** `j`)

Data structures and "template polymorphism"

$$\texttt{list@\{i\}}\ (\texttt{A} : \texttt{Type}_i) \equiv \mathsf{Ind}(X : \texttt{Type}_i)\{X, A \to X \to X\}$$

- By C-Ind:

  $\texttt{list@\{i\}}\ \texttt{A} \preceq \texttt{list@\{j\}}\ \texttt{A}$ (**regardless of** `i` **and** `j`)
- By Conv-Ind:

  $\texttt{list@\{i\}}\ \texttt{A} \simeq \texttt{list@\{j\}}\ \texttt{A}$ (**regardless of** `i` **and** `j`)

Data structures and "template polymorphism"

$$\texttt{list@\{i\}}\ (\texttt{A} : \texttt{Type}_i) \equiv \mathsf{Ind}(X : \texttt{Type}_i)\{X, A \to X \to X\}$$

- By C-Ind:

  $\texttt{list@\{i\}}\ \texttt{A} \preceq \texttt{list@\{j\}}\ \texttt{A}$ (**regardless of** `i` **and** `j`)
- By Conv-Ind:

  $\texttt{list@\{i\}}\ \texttt{A} \simeq \texttt{list@\{j\}}\ \texttt{A}$ (**regardless of** `i` **and** `j`)
- By Conv-Construct:

  $\texttt{nil@\{i\}}\ \texttt{A} \simeq \texttt{nil@\{j\}}\ \texttt{A}$ (**regardless of** `i` **and** `j` as well)

Data structures and "template polymorphism"

$$\texttt{list@\{i\}}\ (\texttt{A} : \texttt{Type}_i) \equiv \mathsf{Ind}(X : \texttt{Type}_i)\{X, A \to X \to X\}$$

- By C-Ind:

  $\texttt{list@\{i\}}\ \texttt{A} \preceq \texttt{list@\{j\}}\ \texttt{A}$ (**regardless of** `i` **and** `j`)
- By Conv-Ind:

  $\texttt{list@\{i\}}\ \texttt{A} \simeq \texttt{list@\{j\}}\ \texttt{A}$ (**regardless of** `i` **and** `j`)
- By Conv-Construct:

  $\texttt{nil@\{i\}}\ \texttt{A} \simeq \texttt{nil@\{j\}}\ \texttt{A}$ (**regardless of** `i` **and** `j` as well)

Models "template-polymorphism", which allows "transparent" copies of inductives (e.g. `list nat` : `Set` while `list` $\texttt{Type}_i$: $\texttt{Type}_{i+1}$ and `prod True True` : `Prop`).

Syntactical models

Cumulative Inductives Types allows to validate cumulativity in syntactical models (Boulier et al. CPP'2017).
Typical, trivial model:

$$[\mathtt{Type}_i] \triangleq (\mathtt{Type}_i \times_{j,\mathtt{Set}} \mathbb{B}, \mathtt{true}) \text{ where } i < j \qquad [\![A]\!] \triangleq [A].1$$

$$\begin{aligned}
[\![\mathtt{Type}_i]\!] \leq [\![\mathtt{Type}_k]\!] \quad &\Leftrightarrow \quad (\mathtt{Type}_i \times_{j,\mathtt{Set}} \mathbb{B}, \mathtt{true}).1 \leq \\
&\qquad (\mathtt{Type}_k \times_{l,\mathtt{Set}} \mathbb{B}, \mathtt{true}).1 \\
&\Leftrightarrow \quad j = l \wedge i = k \qquad \text{(in pCIC)} \\
&\Leftrightarrow \quad i = k \qquad \text{(in pCuIC)}
\end{aligned}$$

```
Inductive TyInterp@{i j | i < j} : Type@{j} :=
  { T : Type@{i}; b : bool }.
```

`TyInterp@{i j}` $\preceq$ `TypInterp@{i' j'}` iff $\mathtt{i} \leq \mathtt{i}'$.

Demo

This is implemented in Coq since version 8.7!

Thanks to Pierre-Marie Pédrot and Gaëtan Gilbert for many improvements on the universe system as well.