Cumulative Inductive Types in Coq

Amin Timany\textsuperscript{1} \quad Matthieu Sozeau\textsuperscript{2}

\textsuperscript{1}imec-Distrinet, KU Leuven, Belgium
\textsuperscript{2}Inria Paris \& IRIF, France

November 14th, 2017

G\textsuperscript{\textregistered}ocal-LAC Days, Nantes, France
What are universes?

Universes are the types of types, e.g:

- nat, bool : Type\(_0\)
- Type\(_0\) : Type\(_1\)
- list : Type\(_0\) → Type\(_0\)
- ∀α : Type\(_0\), list α : Type\(_1\)
- ∀n : nat, {n = 0} + {n ≠ 0} : Type\(_0\)
How are they organised?

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

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

- In higher order dependent type theories, we have a countably infinite hierarchy of universes (types of types):

\[
\text{Type}_0, \text{Type}_1, \text{Type}_2, \ldots
\]

where:

\[
\text{Type}_0 : \text{Type}_1, \text{Type}_1 : \text{Type}_2, \ldots
\]
Universe hierarchy

In higher order dependent type theories, we have a countably infinite hierarchy of universes (types of types):

\[ \text{Type}_0, \text{Type}_1, \text{Type}_2, \ldots \]

where:

\[ \text{Type}_0 : \text{Type}_1, \text{Type}_1 : \text{Type}_2, \ldots \]

Such a system is cumulative if for any type \( T \) and \( i \):

\[ T : \text{Type}_i \Rightarrow T : \text{Type}_{i+1} \]

Example: Predicative Calculus of Inductive Constructions (pCIC), the logic of the proof assistant Coq
Universe cumulativity in pCIC

\[\Gamma \vdash t : A \quad \Gamma \vdash B : s \quad A \leq B \]
\[\Gamma \vdash t : B\]

\[i \leq j\]
\[\text{Type}_i \preceq \text{Type}_j\]

\[A \simeq A' \quad B \preceq B'\]
\[\Pi x : A. \ B \preceq \Pi x : A'. \ B'\]
Universe polymorphism

- pCIC has recently been extended with universe polymorphism
  - Definitions can be polymorphic in universe levels, e.g., categories:

\[
\text{Record Category}_{\{i, j\}} : \text{Type}_{\{\max(i+1, j+1)\}} := \\
\{ \\
\text{Obj} : \text{Type}_{\{i\}}; \\
\text{Hom} : \text{Obj} \to \text{Obj} \to \text{Type}_{\{j\}}; \ldots \}.
\]
Universe polymorphism

- pCIC has recently been extended with universe polymorphism
  - Definitions can 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}; ...
    }.
    ```

- To keep consistency, universe polymorphic definitions come with constraints, e.g., 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:

  \[ k < i \text{ and } l < i \]
Justifying universe polymorphism

- For universe polymorphic inductive types and constants, e.g., `Category`, copies are considered (Sozeau & Tabareau, ITP’14).
- Direct translation from pCIC to CIC + floating universe constraints, itself translatable to CIC with fixed universes.
- With no cumulativity (subtyping), i.e.,
  \[ \text{Category@}\{i \, j\} \preceq \text{Category@}\{k \, l\} \iff i = k \text{ and } j = l \]
Justifying universe polymorphism

- For universe polymorphic inductive types and constants, e.g., Category, copies are considered (Sozeau & Tabareau, ITP’14).
- Direct translation from pCIC to CIC + floating universe constraints, itself translatable to CIC with fixed universes.
- With no cumulativity (subtyping), i.e.,
  \[
  \text{Category} @ \{i, j\} \preceq \text{Category} @ \{k, l\} \iff i = k \quad \text{and} \quad j = l
  \]
- This means \( \text{Cat} @ \{i, j, k, l\} \) is the category of all categories at \( \{k, l\} \) and not lower \(^1\)

\(^1\)There are however categories isomorphic to the categories in lower levels.
Relative size constraints

- Constraints on statements about universe polymorphic inductive definitions restrict to which copies they apply.
- For the category of categories $\textsf{Cat}@\{i,j,k,l\}$ the fact that it has exponentials has constraints $j = k = l$: universe of objects $k = \text{universe of morphisms } l = \text{universe of functors between } k \text{ and } l \text{ categories}.$
Relative size constraints

- Constraints on statements about universe polymorphic inductive definitions restrict to which copies they apply.
- For the category of categories $\text{Cat}@\{i, j, k, l\}$ the fact that it has exponentials has constraints $j = k = l$:
  universe of objects $k =$ universe of morphisms $l =$ universe of functors between $k, l$ categories.
- In particular:

  **Definition** $\text{Type}_\text{Cat}@\{i, j\} :=$

  \[
  \{ | \text{Obj} := \text{Type}@\{j\}; \\
  \quad \text{Hom} := \text{fun } A \to B \Rightarrow A \to B; \ldots | \} : \text{Category}@\{i, j\}.
  \]

  with constraints: $j < i$ is **not** an object of any copy of $\text{Cat}$ with exponentials!
Relative size constraints

- Constraints on statements about universe polymorphic inductive definitions restrict to which copies they apply.
- For the category of categories $\text{Cat}^{i \ j \ k \ l}$ the fact that it has exponentials has constraints $j = k = l$: universe of objects $k = $ universe of morphisms $l = $ universe of functors between $k \ l$ categories.
- In particular:

  \[\text{Definition Type_Cat}^{i \ j} := \{ \text{Obj} := \text{Type}^{i \ j}; \text{Hom} := \text{fun } A \ B \Rightarrow A \rightarrow B; \ldots \} : \text{Category}^{i \ j} \] 

  with constraints: $j < i$ is not an object of any copy of $\text{Cat}$ with exponentials!

- Yoneda embedding can’t be simply defined as the exponential transpose of the $hom$ functor.
Inductive types in pCIC

\[
\text{IND} \quad \begin{array}{c}
A \in Ar(s) \\
\Gamma \vdash A : s' \\
\Gamma, X : A \vdash C_i : s \\
\Gamma \vdash \text{Ind}(X : A)\{C_1, \ldots, C_n\} : A
\end{array}
\]

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

Co(X) is the set of types of the form: \( \Pi \xrightarrow{\bar{x}} \bar{M}. \ X \xrightarrow{\bar{m}} \)
Inductive types in pCIC

\[ \text{IND} \]
\[ A \in Ar(s) \quad \Gamma \vdash A : s' \quad \Gamma, X : A \vdash C_i : s \quad C_i \in Co(X) \]
\[ \Gamma \vdash \text{Ind}(X : A)\{C_1, \ldots, C_n\} : A \]

\( Ar(s) \) is the set of types of the form: \( \Pi \xrightarrow{\chi} M. \ s \)

\( Co(X) \) is the set of types of the form: \( \Pi \xrightarrow{\chi} M. \ X \xrightarrow{m} \)

No Parameters (\( T \) in \( \text{vec} \ T \ n \)) are considered in this rule.

Inductive \( \text{vec} \ (T : \text{Type}) : \text{nat} \rightarrow \text{Type} := \)  
\[ \text{nil} : \text{vec} \ T \ 0 \]
\[ \text{cons} : \forall n, T \rightarrow \text{vec} \ T \ n \rightarrow \text{vec} \ T \ (S \ n). \]
Inductive types in pCIC

\[
\text{IND} \\
\begin{array}{llll}
A \in Ar(s) & \Gamma \vdash A : s' & \Gamma, X : A \vdash C_i : s & C_i \in Co(X) \\
\hline
\Gamma \vdash \text{Ind}(X : A)\{C_1, \ldots, C_n\} : A
\end{array}
\]

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

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

No Parameters (\(T\) in \(\text{vec} T\ n\)) are considered in this rule.

Inductive \(\text{vec} (T : \text{Type}) : \text{nat} \rightarrow \text{Type} := \text{nil} : \text{vec} T\ 0\)
| \(\text{cons} : \text{forall} n, T \rightarrow \text{vec} T\ n \rightarrow \text{vec} T\ (S\ n)\).
Predicative Calculus of Cumulative Inductive Types (pCuIC)

\[ \text{C-IND} \]

\[ l \equiv (\text{Ind}(X : \Pi \vec{x} : \vec{N}. s)\{\Pi \vec{x}_1 : \vec{M}_1. X \ \vec{m}_1, \ldots, \Pi \vec{x}_n : \vec{M}_n. X \ \vec{m}_n\}) \]

\[ l' \equiv (\text{Ind}(X : \Pi \vec{x} : \vec{N}'. s')\{\Pi \vec{x}_1 : \vec{M}_1'. X \ \vec{m}_1', \ldots, \Pi \vec{x}_n : \vec{M}_n'. X \ \vec{m}_n'\}) \]

\[
\forall i. N_i \preceq N'_i \quad \forall i, j. (M_i)_j \preceq (M'_i)_j \\
\text{length}(\vec{m}) = \text{length}(\vec{x}) \quad \forall i. X \ \vec{m}_i \simeq X \ \vec{m}'_i
\]

\[ \vdash l \ \vec{m} \preceq l' \ \vec{m} \]
Predicative Calculus of Cumulative Inductive Types (pCuIC)

\[
\begin{align*}
C-IND & \quad I \equiv (\text{Ind}(X : \prod \vec{x} : \vec{N}. s)\{\prod \vec{x}_1 : \vec{M}_1. X \ m_1, \ldots, \prod \vec{x}_n : \vec{M}_n. X \ m_n\}) \\
& \quad I' \equiv (\text{Ind}(X : \prod \vec{x} : \vec{N}'. s')\{\prod \vec{x}_1 : \vec{M}'_1. X \ m'_1, \ldots, \prod \vec{x}_n : \vec{M}'_n. X \ m'_n\}) \\
& \quad \forall i. \ N_i \leq N'_i \quad \forall i, j. (M_i)_j \leq (M'_i)_j \\
& \quad \text{length}(\vec{m}) = \text{length}(\vec{x}) \quad \forall i. \ X \ m_i \simeq X \ m'_i \\
\hline
& \quad I \ m \preceq I' \ m
\end{align*}
\]
Predicative Calculus of **Cumulative Inductive Types** (pCuIC)

\[ C-I\text{ND} \]

\[ I ≡ (\text{Ind}(X : \prod \vec{x} : \vec{N}. \ s)\{\prod \vec{x}_1 : \vec{M}_1. \ X \ m_1, \ldots, \prod \vec{x}_n : \vec{M}_n. \ X \ m_n\}) \]

\[ I' ≡ (\text{Ind}(X : \prod \vec{x} : \vec{N}'. \ s')\{\prod \vec{x}_1 : \vec{M}_1'. \ X \ m_1', \ldots, \prod \vec{x}_n : \vec{M}_n'. \ X \ m_n'\}) \]

\[ \forall i. \ N_i \preceq N'_i \quad \forall i, j. \ (M_i)_j \preceq (M'_i)_j \]

\[ \text{length}(\vec{m}) = \text{length}(\vec{x}) \quad \forall i. \ X \ m_i \simeq X \ m'_i \]

\[ I \vec{m} \preceq I' \vec{m} \]
Predicative Calculus of Cumulative Inductive Types (pCuIC)

\[
\text{C-IND}
\]

\[
I \equiv (\text{Ind}(X : \prod \vec{x} : \vec{N}. s)\{\prod \vec{x}_1 : \vec{M}_1. X \ m_1, \ldots, \prod \vec{x}_n : \vec{M}_n. X \ m_n\})
\]

\[
I' \equiv (\text{Ind}(X : \prod \vec{x} : \vec{N}'. s')\{\prod \vec{x}_1 : \vec{M}'_1. X \ m'_1, \ldots, \prod \vec{x}_n : \vec{M}'_n. X \ m'_n\})
\]

\[
\forall i. \ N_i \preceq N'_i \quad \forall i, j. (M_i)_j \preceq (M'_i)_j
\]

\[
\text{length}(\vec{m}) = \text{length}(\vec{x}) \quad \forall i. \ X \ \vec{m}_i \simeq X \ \vec{m}'_i
\]

\[
I \ \vec{m} \preceq I' \ \vec{m}
\]
Predicative Calculus of Cumulative Inductive Types (pCuIC)

\[ \text{C-IND} \]

\[ I \equiv (\text{Ind}(X : \Pi \vec{x} : \vec{N}. s)\{\Pi \vec{x}_1 : \vec{M}_1. X \vec{m}_1, \ldots, \Pi \vec{x}_n : \vec{M}_n. X \vec{m}_n\}) \]

\[ I' \equiv (\text{Ind}(X : \Pi \vec{x} : \vec{N}'. s')\{\Pi \vec{x}_1 : \vec{M}'_1. X \vec{m}'_1, \ldots, \Pi \vec{x}_n : \vec{M}'_n. X \vec{m}'_n\}) \]

\[ \forall i. N_i \preceq N'_i \quad \forall i, j. (M_i)_j \preceq (M'_i)_j \]

\[ \text{length}(\vec{m}) = \text{length}(\vec{x}) \quad \forall i. X \vec{m}_i \simeq X \vec{m}'_i \]

\[ I \vec{m} \preceq I' \vec{m} \]
C-IND

\[ I \equiv (\text{Ind}(X : \prod \vec{x} : \vec{N}. \ s)\{\prod \vec{x}_1 : \vec{M}_1. X \ m_1, \ldots, \prod \vec{x}_n : \vec{M}_n. X \ m_n\}) \]

\[ I' \equiv (\text{Ind}(X : \prod \vec{x} : \vec{N}'. s')\{\prod \vec{x}_1 : \vec{M}'_1. X \ m'_1, \ldots, \prod \vec{x}_n : \vec{M}'_n. X \ m'_n\}) \]

\[ \forall i. N_i \preceq N'_i \quad \forall i, j. (M_i)_j \preceq (M'_i)_j \]

\[ \text{length}(\vec{m}) = \text{length}(\vec{x}) \quad \forall i. X \ m_i \simeq X \ m'_i \]

\[ I \vec{m} \preceq I' \vec{m} \]
Predicative Calculus of Cumulative Inductive Types (pCuIC)

\[ C\text{-}IND \]

\[ I \equiv (\text{Ind}(X : \Pi \vec{x} : \vec{N}. s)\{\Pi \vec{x}_1 : \vec{M}_1. X \vec{m}_1, \ldots, \Pi \vec{x}_n : \vec{M}_n. X \vec{m}_n\}) \]

\[ I' \equiv (\text{Ind}(X : \Pi \vec{x} : \vec{N}'. s')\{\Pi \vec{x}_1 : \vec{M}_1'. X \vec{m}_1', \ldots, \Pi \vec{x}_n : \vec{M}_n'. X \vec{m}_n'\}) \]

\[ \forall i. N_i \preceq N'_i \quad \forall i, j. (M_i)_j \preceq (M'_i)_j \]

\[ \text{length}(\vec{m}) = \text{length}(\vec{x}) \quad \forall i. X \vec{m}_i \simeq X \vec{m}'_i \]

\[ I \vec{m} \preceq I' \vec{m} \]


▶ Example:

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

▶ By C-IND:

\[ i \leq k \text{ and } j \leq l \Rightarrow \text{Category}@\{i \ j\} \preceq \text{Category}@\{k \ l\} \]
Predicative Calculus of Cumulative Inductive Types (pCuIC)

\[ C\text{-IND} \]

\[ I \equiv (\text{Ind}(X : \Pi \vec{x} : \vec{N}. s)\{\Pi \vec{x}_1 : \vec{M}_1. X \vec{m}_1, \ldots, \Pi \vec{x}_n : \vec{M}_n. X \vec{m}_n\}) \]

\[ I' \equiv (\text{Ind}(X : \Pi \vec{x} : \vec{N}'. s')\{\Pi \vec{x}_1 : \vec{M}'_1. X \vec{m}'_1, \ldots, \Pi \vec{x}_n : \vec{M}'_n. X \vec{m}'_n\}) \]

\[ \forall i. N_i \preceq N'_i \quad \forall i, j. (M_i)_j \preceq (M'_i)_j \]

\[ \text{length}(\vec{m}) = \text{length}(\vec{x}) \quad \forall i. X \vec{m}_i \simeq X \vec{m}'_i \]

\[ I \vec{m} \preceq I' \vec{m} \]

- **Example:**

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

- **By C-IND:**

  \[ i \leq k \text{ and } j \leq l \Rightarrow \text{Category@}\{i, j\} \preceq \text{Category@}\{k, l\} \]

- **Notice C-IND does not consider parameters or sort of the inductive type**
For \( \text{Cat@}\{i \ j \ k \ l\} \) with exponentials we had the constraints:
\[
j = k = l
\]

- Type\(_\text{Cat@}\{i' \ j'\} \) we had the constraint: \( j' < i' \)
- Now Type\(_\text{Cat@}\{i' \ j'\} : \text{Obj} \text{Cat@}\{i \ j \ k \ l\} \) just imposes the constraint: \( i' \leq k \land j' \leq l \), which is consistent.
Data structures and “template polymorphism”

\[ \text{list@}\{i\} (A : \text{Type}_i) \equiv \text{Ind}(X : \text{Type}_i)\{X, A \to X \to X\} \]

By C-IND:

\[ \text{list@}\{i\} A \preceq \text{list@}\{j\} A \]
Data structures and “template polymorphism”

\[
\text{list@}\{i\} \ (A : \text{Type}_i) \equiv \text{Ind} (X : \text{Type}_i) \{ X, A \to X \to X \}
\]

- By \text{C-IND}:
  \[
  \text{list@}\{i\} \ A \preceq \text{list@}\{j\} \ A \quad (\text{regardless of } i \text{ and } j)
  \]
Data structures and “template polymorphism”

\[
\text{list@}\{i\} (A : \text{Type}_i) \equiv \text{Ind}(X : \text{Type}_i)\{X, A \rightarrow X \rightarrow X\}
\]

- By C-**IND**: 
  \[
  \text{list@}\{i\} A \preceq \text{list@}\{j\} A \quad (\text{regardless of } i \text{ and } j)
  \]

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

  **Conv-IND**

  \[
  \begin{align*}
  & I \vec{m} \preceq I' \vec{m} \quad I' \vec{m} \preceq I \vec{m} \\
  \hline
  & I \vec{m} \simeq I' \vec{m}
  \end{align*}
  \]

- Properly models “template-polymorphism”, which allows “transparent” copies of inductives (e.g. \( \text{list nat} : \text{Set} \) while \( \text{list Type} i : \text{Type}_i + 1 \)).
Data structures and “template polymorphism”

\[
\text{list@\{i\} (A : \text{Type}_i) \equiv \text{Ind}(X : \text{Type}_i)\{X, A \rightarrow X \rightarrow X\}}
\]

- By C-\text{IND}:
  \[
  \text{list@\{i\} A \preceq \text{list@\{j\} A} \quad \text{(regardless of } i \text{ and } j)\]

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

\[
\begin{align*}
\text{Conv-IND} & \\
I \vec{m} \preceq I' \vec{m} & \quad I' \vec{m} \preceq I \vec{m} \\
\hline \\
I \vec{m} & \simeq I' \vec{m}
\end{align*}
\]

- \( i = k \) and \( j = l \) \( \Rightarrow \) \( \text{Category@\{i j\} \simeq \text{Category@\{k l\}}\)

- \( \text{list@\{i\} A \simeq \text{list@\{j\} A} \quad \text{(regardless of } i \text{ and } j)\)

- \( \text{nil@\{i\} A \simeq \text{nil@\{j\} A} \quad \text{(regardless of } i \text{ and } j \text{ as well)}\)

- Properly models “template-polymorphism”, which allows “transparent” copies of inductives (e.g. \text{list} \text{nat} : \text{Set} while \text{list Type}_i : \text{Type}_{i+1}).
This is implemented in Coq 8.7!

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

We construct a set theoretic model for pCuIC $\mathcal{M} : Terms_{pCuIC} \rightarrow ZFC$: ZFC with suitable axioms, e.g., inaccessible cardinals, to model pCuIC universes.

- Based on a modification of Werner & Lee’s model which assumed Strong Normalization.
- For subtyping $A \preceq B$ we have $\mathcal{M}[A] \subseteq \mathcal{M}[B]$.
- Inductive types interpreted using least fixpoints of monotone\textsuperscript{2} functions, eliminators instead of match+fix+guard condition.
- This justifies both $C$-\textsc{Ind} and $Conv$-$\textsc{Ind}$ and even conversion of constructors applied to parameters.

\textsuperscript{2}Due to strict positivity condition
The End

Thanks