First Steps Towards Cumulative Inductive Types in CIC

Amin Timany

Bart Jacobs

iMinds-Distrinet KU Leuven

ICTAC'15 – October 31, 2015

Outline

- In higher order dependent type theories:
  - Types are also terms and hence have a type

- In higher order dependent type theories:
  - Types are also terms and hence have a type
  - Type of all types, as it should be the type of itself, leads to paradoxes, like Russell's paradox in set theory

- In higher order dependent type theories:
  - Types are also terms and hence have a type
  - Type of all types, as it should be the type of itself, leads to paradoxes, like Russell's paradox in set theory
  - Thus, we have a countably infinite hierarchy of universes (types of types):

$$\texttt{Type}_0, \texttt{Type}_1, \texttt{Type}_2, \ldots$$

- In higher order dependent type theories:
  - Types are also terms and hence have a type
  - Type of all types, as it should be the type of itself, leads to paradoxes, like Russell's paradox in set theory
  - Thus, we have a countably infinite hierarchy of universes (types of types):

$$\mathtt{Type}_0, \mathtt{Type}_1, \mathtt{Type}_2, \ldots$$

    where:

$$\mathtt{Type}_0 : \mathtt{Type}_1, \mathtt{Type}_1 : \mathtt{Type}_2, \ldots$$

- In higher order dependent type theories:
  - Types are also terms and hence have a type
  - Type of all types, as it should be the type of itself, leads to paradoxes, like Russell's paradox in set theory
  - Thus, we have a countably infinite hierarchy of universes (types of types):

$$\mathtt{Type}_0, \mathtt{Type}_1, \mathtt{Type}_2, \ldots$$

    where:

$$\mathtt{Type}_0 : \mathtt{Type}_1, \mathtt{Type}_1 : \mathtt{Type}_2, \ldots$$

- Such a system is cumulative if for any type T and i:

$$T : \mathtt{Type}_i \Rightarrow T : \mathtt{Type}_{i+1}$$

- In higher order dependent type theories:
  - Types are also terms and hence have a type
  - Type of all types, as it should be the type of itself, leads to paradoxes, like Russell's paradox in set theory
  - Thus, we have a countably infinite hierarchy of universes (types of types):

    $$\mathtt{Type}_0, \mathtt{Type}_1, \mathtt{Type}_2, \ldots$$

    where:

    $$\mathtt{Type}_0 : \mathtt{Type}_1, \mathtt{Type}_1 : \mathtt{Type}_2, \ldots$$

- Such a system is cumulative if for any type T and i:

$$T : \mathtt{Type}_i \Rightarrow T : \mathtt{Type}_{i+1}$$

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

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

- 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};
    ⋮
  }.
```

- 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 consistent, universe polymorphic definitions come with constraints, e.g., category of categories:

- 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 consistent, 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:

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

- For universe polymorphic inductive types, e.g., `Category`, copies are considered

- For universe polymorphic inductive types, e.g., `Category`, copies are considered
- With no cumulativity, i.e.,

- For universe polymorphic inductive types, e.g., `Category`, copies are considered
- With no cumulativity, i.e.,
  `C : Category@{i j}` and `C : Category@{k l}` implies $\mathtt{i} = \mathtt{k}$ and $\mathtt{j} = \mathtt{l}$

- For universe polymorphic inductive types, e.g., `Category`, copies are considered
- With no cumulativity, i.e.,
  `C : Category@{i j}` and `C : Category@{k l}` implies `i` = `k` and `j` = `l`
- This means `Cat@{i j k l}` is the category of all categories at {`k l`} and *not* lower

- For universe polymorphic inductive types, e.g., `Category`, copies are considered
- With no cumulativity, i.e.,
  `C : Category@{i j}` and `C : Category@{k l}` implies `i` = `k` and `j` = `l`
- This means `Cat@{i j k l}` is the category of all categories at `{k l}` and *not* lower
- Constraints on statements about universe polymorphic inductive definitions restrict to which copies they apply

- For universe polymorphic inductive types, e.g., `Category`, copies are considered
- With no cumulativity, i.e.,
  `C : Category@{i j}` and `C : Category@{k l}` implies `i` = `k` and `j` = `l`
- This means `Cat@{i j k l}` is the category of all categories at `{k l}` and *not* lower
- Constraints on statements about universe polymorphic inductive definitions restrict to which copies they apply
- For `Cat@{i j k l}` the fact that it has exponentials has constraints `j` = `k` = `l`

- For universe polymorphic inductive types, e.g., `Category`, copies are considered
- With no cumulativity, i.e.,
  `C : Category@{i j}` and `C : Category@{k l}` implies `i` $=$ `k` and `j` $=$ `l`
- This means `Cat@{i j k l}` is the category of all categories at `{k l}` and *not* lower
- Constraints on statements about universe polymorphic inductive definitions restrict to which copies they apply
- For `Cat@{i j k l}` the fact that it has exponentials has constraints `j` $=$ `k` $=$ `l`
- In particular:

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

with constraints: $j < i$

- For universe polymorphic inductive types, e.g., `Category`, copies are considered
- With no cumulativity, i.e.,
  `C : Category@{i j}` and `C : Category@{k l}` implies `i` = `k` and `j` = `l`
- This means `Cat@{i j k l}` is the category of all categories at `{k l}` and *not* lower
- Constraints on statements about universe polymorphic inductive definitions restrict to which copies they apply
- For `Cat@{i j k l}` the fact that it has exponentials has constraints `j` = `k` = `l`
- 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!

- For universe polymorphic inductive types, e.g., `Category`, copies are considered
- With no cumulativity, i.e.,
  `C : Category@{i j}` and `C : Category@{k l}` implies `i` $=$ `k` and `j` $=$ `l`
- This means `Cat@{i j k l}` is the category of all categories at `{k l}` and *not* lower
- Constraints on statements about universe polymorphic inductive definitions restrict to which copies they apply
- For `Cat@{i j k l}` the fact that it has exponentials has constraints `j` $=$ `k` $=$ `l`
- 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

- Not restricted to categories:

- Not restricted to categories:
  - Consider inductive representation of ensembles:

```
Inductive Ens@{i} : Type@{i+1} :=
   ens@{i} : forall (A : Type@{i}), (A → Ens@{i}) → Ens@{i}
.
```

- Not restricted to categories:
  - Consider inductive representation of ensembles:

```
Inductive Ens@{i} : Type@{i+1} :=
   ens@{i} : forall (A : Type@{i}), (A → Ens@{i}) → Ens@{i}
.
```

  - Examples:

```
empty := ens@{0} Empty (Empty_rect Ens@{i})
```

- Not restricted to categories:
  - Consider inductive representation of ensembles:

```
Inductive Ens@{i} : Type@{i+1} :=
   ens@{i} : forall (A : Type@{i}), (A → Ens@{i}) → Ens@{i}
.
```

  - Examples:

```
empty := ens@{0} Empty (Empty_rect Ens@{i})

union (ens@{i} A f) (ens@{i} B g) := ens@{i} (A + B) (f + g)
```

- Not restricted to categories:
  - Consider inductive representation of ensembles:

```
Inductive Ens@{i} : Type@{i+1} :=
   ens@{i} : forall (A : Type@{i}), (A → Ens@{i}) → Ens@{i}
.
```

  - Examples:

```
empty := ens@{0} Empty (Empty_rect Ens@{i})

union (ens@{i} A f) (ens@{i} B g) := ens@{i} (A + B) (f + g)

intersection (ens@{i} A f) (ens@{i} B g) := ens@{i} (A × B) (f × g)
```

- Not restricted to categories:
  - Consider inductive representation of ensembles:

```
Inductive Ens@{i} : Type@{i+1} :=
   ens@{i} : forall (A : Type@{i}), (A → Ens@{i}) → Ens@{i}
.
```

  - Examples:

```
empty := ens@{0} Empty (Empty_rect Ens@{i})

union (ens@{i} A f) (ens@{i} B g) := ens@{i} (A + B) (f + g)

intersection (ens@{i} A f) (ens@{i} B g) := ens@{i} (A × B) (f × g)
```

  - Ensemble of small ensembles can't be directly formed:

```
ens@{j+1} Ens@{j} (fun x : Ens@{j} ⇒ x)
```

- Not restricted to categories:
  - Consider inductive representation of ensembles:

```
Inductive Ens@{i} : Type@{i+1} :=
   ens@{i} : forall (A : Type@{i}), (A → Ens@{i}) → Ens@{i}
.
```

  - Examples:

```
empty := ens@{0} Empty (Empty_rect Ens@{i})

union (ens@{i} A f) (ens@{i} B g) := ens@{i} (A + B) (f + g)

intersection (ens@{i} A f) (ens@{i} B g) := ens@{i} (A × B) (f × g)
```

  - Ensemble of small ensembles can't be directly formed:

```
ens@{j+1} Ens@{j} (fun x : Ens@{j} ⇒ x)
```

- Can be solved using liftings, e.g.,

- Not restricted to categories:
  - Consider inductive representation of ensembles:

```
Inductive Ens@{i} : Type@{i+1} :=
   ens@{i} : forall (A : Type@{i}), (A → Ens@{i}) → Ens@{i}
.
```

  - Examples:

```
empty := ens@{0} Empty (Empty_rect Ens@{i})

union (ens@{i} A f) (ens@{i} B g) := ens@{i} (A + B) (f + g)

intersection (ens@{i} A f) (ens@{i} B g) := ens@{i} (A × B) (f × g)
```

  - Ensemble of small ensembles can't be directly formed:

```
ens@{j+1} Ens@{j} (fun x : Ens@{j} ⇒ x)
```

- Can be solved using liftings, e.g.,

```
ens_lift@{i k} : Ens@{i} → Ens@{k}
```

with the side condition: $i \leq k$.

- Not restricted to categories:
  - Consider inductive representation of ensembles:

```
Inductive Ens@{i} : Type@{i+1} :=
   ens@{i} : forall (A : Type@{i}), (A → Ens@{i}) → Ens@{i}
.
```

  - Examples:

```
empty := ens@{0} Empty (Empty_rect Ens@{i})

union (ens@{i} A f) (ens@{i} B g) := ens@{i} (A + B) (f + g)

intersection (ens@{i} A f) (ens@{i} B g) := ens@{i} (A × B) (f × g)
```

  - Ensemble of small ensembles can't be directly formed:

```
ens@{j+1} Ens@{j} (fun x : Ens@{j} ⇒ x)
```

- Can be solved using liftings, e.g.,

```
ens_lift@{i k} : Ens@{i} → Ens@{k}
```

  with the side condition: $\mathtt{i} \leq \mathtt{k}$.
- Problem: `e` and `ens_lift e` are *not* necessarily the same

- Not restricted to categories:
  - Consider inductive representation of ensembles:

```
Inductive Ens@{i} : Type@{i+1} :=
   ens@{i} : forall (A : Type@{i}), (A → Ens@{i}) → Ens@{i}
.
```

  - Examples:

```
empty := ens@{0} Empty (Empty_rect Ens@{i})

union (ens@{i} A f) (ens@{i} B g) := ens@{i} (A + B) (f + g)

intersection (ens@{i} A f) (ens@{i} B g) := ens@{i} (A × B) (f × g)
```

  - Ensemble of small ensembles can't be directly formed:

```
ens@{j+1} Ens@{j} (fun x : Ens@{j} ⇒ x)
```

- Can be solved using liftings, e.g.,

  `ens_lift@{i k} : Ens@{i} → Ens@{k}`

  with the side condition: $\mathtt{i} \leq \mathtt{k}$.
- Problem: `e` and `ens_lift e` are *not* necessarily the same
- Any statement about `e` is not usable with `ens_lift e` and needs to be proven separately

Outline

- Some (*simplified*) typing rules of pCIC:

- Some (*simplified*) typing rules of pCIC:

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

- Some (*simplified*) typing rules of pCIC:

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

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\lambda x : A.\ t) : (\Pi x : A.\ B)} \quad (\textsc{Lam})$$

- Some (*simplified*) typing rules of pCIC:

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

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\lambda x : A.\ t) : (\Pi x : A.\ B)} \quad (\textsc{Lam})$$

$$\frac{\Gamma \vdash t : (\Pi x : A.B) \quad \Gamma \vdash t' : A}{\Gamma \vdash (t\ t') : B[t'/x]} \quad (\textsc{App})$$

- Some (*simplified*) typing rules of pCIC:

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

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\lambda x : A.\ t) : (\Pi x : A.\ B)} \quad (\textsc{Lam})$$

$$\frac{\Gamma \vdash t : (\Pi x : A.B) \quad \Gamma \vdash t' : A}{\Gamma \vdash (t\ t') : B[t'/x]} \quad (\textsc{App})$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash B : s \quad A \preceq B}{\Gamma \vdash t : B} \quad (\textsc{Conv})$$

- Some (*simplified*) typing rules of pCIC:

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

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\lambda x : A.\ t) : (\Pi x : A.\ B)} \quad (\textsc{Lam})$$

$$\frac{\Gamma \vdash t : (\Pi x : A.B) \quad \Gamma \vdash t' : A}{\Gamma \vdash (t\ t') : B[t'/x]} \quad (\textsc{App})$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash B : s \quad A \preceq B}{\Gamma \vdash t : B} \quad (\textsc{Conv})$$

$$\frac{A \in Ar(s) \quad \Gamma \vdash A : s' \quad (\Gamma, X : A \vdash C_i : s \quad C_i \in Co(X)\ \forall 1 \leq i \leq n)}{\Gamma \vdash \mathsf{Ind}(X : A)\{C_1, \ldots, C_n\} : A} \quad (\textsc{Ind})$$

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

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

- Examples:
  - Prod:

$$\frac{A : \mathtt{Type}_i, n : nat \vdash \mathbb{V}\mathtt{ect}_{A,n} : \mathtt{Type}_i \quad A : \mathtt{Type}_i \vdash nat : \mathtt{Type}_0}{A : \mathtt{Type}_i \vdash (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n}) : \mathtt{Type}_i}$$

- Examples:
  - Prod:

$$\frac{A : \mathtt{Type}_i, n : nat \vdash \mathbb{V}\mathtt{ect}_{A,n} : \mathtt{Type}_i \quad A : \mathtt{Type}_i \vdash nat : \mathtt{Type}_0}{A : \mathtt{Type}_i \vdash (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n}) : \mathtt{Type}_i}$$

  - Lam:

$$\frac{A : \mathtt{Type}_i, n : nat \vdash t : \mathbb{V}\mathtt{ect}_{A,n}}{A : \mathtt{Type}_i \vdash (\lambda n : nat.\ t) : (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n})}$$

- Examples:
  - PROD:

$$\frac{A : \mathtt{Type}_i, n : nat \vdash \mathbb{V}\mathtt{ect}_{A,n} : \mathtt{Type}_i \quad A : \mathtt{Type}_i \vdash nat : \mathtt{Type}_0}{A : \mathtt{Type}_i \vdash (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n}) : \mathtt{Type}_i}$$

  - LAM:

$$\frac{A : \mathtt{Type}_i, n : nat \vdash t : \mathbb{V}\mathtt{ect}_{A,n}}{A : \mathtt{Type}_i \vdash (\lambda n : nat.\ t) : (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n})}$$

  - APP:

$$\frac{A : \mathtt{Type}_i \vdash f : (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n}) \quad A : \mathtt{Type}_i \vdash x : nat}{A : \mathtt{Type}_i \vdash f\ x : \mathbb{V}\mathtt{ect}_{A,x}}$$

- Examples:
  - Prod:

$$\frac{A : \mathtt{Type}_i, n : nat \vdash \mathbb{V}\mathtt{ect}_{A,n} : \mathtt{Type}_i \quad A : \mathtt{Type}_i \vdash nat : \mathtt{Type}_0}{A : \mathtt{Type}_i \vdash (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n}) : \mathtt{Type}_i}$$

  - Lam:

$$\frac{A : \mathtt{Type}_i, n : nat \vdash t : \mathbb{V}\mathtt{ect}_{A,n}}{A : \mathtt{Type}_i \vdash (\lambda n : nat.\ t) : (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n})}$$

  - App:

$$\frac{A : \mathtt{Type}_i \vdash f : (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n}) \quad A : \mathtt{Type}_i \vdash x : nat}{A : \mathtt{Type}_i \vdash f\ x : \mathbb{V}\mathtt{ect}_{A,x}}$$

  - Ind:

$$\cdot \vdash \mathsf{Ind}(nat : \mathtt{Type}_0)\{nat, nat \to nat\}$$

- Examples:
  - PROD:

$$\frac{A : \mathtt{Type}_i, n : nat \vdash \mathbb{V}\mathtt{ect}_{A,n} : \mathtt{Type}_i \quad A : \mathtt{Type}_i \vdash nat : \mathtt{Type}_0}{A : \mathtt{Type}_i \vdash (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n}) : \mathtt{Type}_i}$$

  - LAM:

$$\frac{A : \mathtt{Type}_i, n : nat \vdash t : \mathbb{V}\mathtt{ect}_{A,n}}{A : \mathtt{Type}_i \vdash (\lambda n : nat.\ t) : (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n})}$$

  - APP:

$$\frac{A : \mathtt{Type}_i \vdash f : (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n}) \quad A : \mathtt{Type}_i \vdash x : nat}{A : \mathtt{Type}_i \vdash f\ x : \mathbb{V}\mathtt{ect}_{A,x}}$$

  - IND:

$$\cdot \vdash \mathsf{Ind}(nat : \mathtt{Type}_0)\{nat, nat \to nat\}$$

$$A : \mathtt{Type}_i \vdash \mathsf{Ind}(Vect_A : nat \to \mathtt{Type}_i)\{Vect_A\ 0, \Pi n : nat.\ A \to Vect_A\ n \to Vect_A\ (S\ n)\}$$

- Examples:
  - PROD:

$$\frac{A : \texttt{Type}_i, n : nat \vdash \mathbb{V}\mathtt{ect}_{A,n} : \texttt{Type}_i \quad A : \texttt{Type}_i \vdash nat : \texttt{Type}_0}{A : \texttt{Type}_i \vdash (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n}) : \texttt{Type}_i}$$

  - LAM:

$$\frac{A : \texttt{Type}_i, n : nat \vdash t : \mathbb{V}\mathtt{ect}_{A,n}}{A : \texttt{Type}_i \vdash (\lambda n : nat.\ t) : (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n})}$$

  - APP:

$$\frac{A : \texttt{Type}_i \vdash f : (\Pi n : nat.\ \mathbb{V}\mathtt{ect}_{A,n}) \quad A : \texttt{Type}_i \vdash x : nat}{A : \texttt{Type}_i \vdash f\ x : \mathbb{V}\mathtt{ect}_{A,x}}$$

  - IND:

$$\cdot \vdash \mathsf{Ind}(nat : \texttt{Type}_0)\{nat, nat \to nat\}$$

$$A : \texttt{Type}_i \vdash \underbrace{\mathsf{Ind}(Vect_A : nat \to \texttt{Type}_i)\{Vect_A\ 0, \Pi n : nat.\ A \to Vect_A\ n \to Vect_A\ (S\ n)\}}_{I}$$

$$\mathbb{V}\mathtt{ect}_{A,n} \triangleq I\ n$$

- Conversion/cumulativity rules of pCIC:

- Conversion/cumulativity rules of pCIC:

$$\frac{i \leq j}{\mathtt{Type}_i \preceq \mathtt{Type}_j} \quad (\textsc{C-Type})$$

- Conversion/cumulativity rules of pCIC:

$$\frac{i \leq j}{\texttt{Type}_i \preceq \texttt{Type}_j} \quad (\textsc{C-Type})$$

$$\frac{A \simeq A' \quad B \preceq B'}{\Pi x : A.\ B \preceq \Pi x : A'.\ B'} \quad (\textsc{C-Prod})$$

Outline

- Predicative Calculus of Cumulative Inductive Types (pCuIC):

- Predicative Calculus of Cumulative Inductive Types (pCuIC):
- pCuIC is pCIC + C-Ind rule:

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

- Predicative Calculus of Cumulative Inductive Types (pCuIC):
- pCuIC is pCIC + C-Ind rule:

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

- Predicative Calculus of Cumulative Inductive Types (pCuIC):
- pCuIC is pCIC + C-Ind rule:

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

- Predicative Calculus of Cumulative Inductive Types (pCuIC):
- pCuIC is pCIC + C-IND rule:

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

- Predicative Calculus of Cumulative Inductive Types (pCuIC):
- pCuIC is pCIC + C-Ind rule:

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

- Predicative Calculus of Cumulative Inductive Types (pCuIC):
- pCuIC is pCIC + C-Ind rule:

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

- Predicative Calculus of Cumulative Inductive Types (pCuIC):
- pCuIC is pCIC + C-Ind rule:

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

- Example:

$\texttt{Category@\{i j\}} \equiv \mathsf{Ind}(X : \texttt{Type}_{max(i+1,j+1)})\{\Pi o : \texttt{Type}_i.\Pi h : o \to o \to \texttt{Type}_j.N\}$
where i and j don't appear in term N

- Predicative Calculus of Cumulative Inductive Types (pCuIC):
- pCuIC is pCIC + C-Ind rule:

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

- Example:

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

where i and j don't appear in term N

- By C-Ind:

$$\texttt{Type}_i \preceq \texttt{Type}_k \text{ and } \texttt{Type}_j \preceq \texttt{Type}_l \Rightarrow \texttt{Category@\{i j\}} \preceq \texttt{Category@\{k l\}}$$

- Predicative Calculus of Cumulative Inductive Types (pCuIC):
- pCuIC is pCIC + C-IND rule:

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

- Example:

$\texttt{Category@\{i j\}} \equiv \mathsf{Ind}(X : \texttt{Type}_{max(i+1,j+1)})\{\Pi o : \texttt{Type}_i.\Pi h : o \to o \to \texttt{Type}_j.N\}$
where i and j don't appear in term N

- By C-IND:

$$\texttt{Type}_i \preceq \texttt{Type}_k \text{ and } \texttt{Type}_j \preceq \texttt{Type}_l \Rightarrow \texttt{Category@\{i j\}} \preceq \texttt{Category@\{k l\}}$$

Also:

$$\texttt{Ens@\{i\}} \equiv \mathsf{Ind}(X : \texttt{Type}_{i+1})\{\Pi A : \texttt{Type}_i.(A \to X) \to X\}$$

- Predicative Calculus of Cumulative Inductive Types (pCuIC):
- pCuIC is pCIC + C-Ind rule:

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

- Example:

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

where i and j don't appear in term N

- By C-Ind:

$$\texttt{Type}_i \preceq \texttt{Type}_k \text{ and } \texttt{Type}_j \preceq \texttt{Type}_l \Rightarrow \texttt{Category@\{i j\}} \preceq \texttt{Category@\{k l\}}$$

Also:

$$\texttt{Ens@\{i\}} \equiv \mathsf{Ind}(X : \texttt{Type}_{i+1})\{\Pi A : \texttt{Type}_i.(A \to X) \to X\}$$

$$\texttt{Type}_i \preceq \texttt{Type}_k \Rightarrow \texttt{Ens@\{i\}} \preceq \texttt{Ens@\{k\}}$$

Conjecture

pCuIC has the following properties:

1. *Church-Rosser property*
2. *Strong normalization*
3. *Context Validity*
4. *Typing Validity*
5. *Subject Reduction*

Conjecture

pCuIC has the following properties:

1. *Church-Rosser property*
2. *Strong normalization*
3. *Context Validity*
4. *Typing Validity*
5. *Subject Reduction*

Conjecture

Let $\Gamma \vdash_{\mathsf{pCIC}} T : s$ *be a pCIC type such that* $\Gamma \vdash_{\mathsf{pCuIC}} t : T$. *Then there exists a term* t' *such that* $\Gamma \vdash_{\mathsf{pCIC}} t' : T$.

Conjecture

pCuIC has the following properties:

1. *Church-Rosser property*
2. *Strong normalization*
3. *Context Validity*
4. *Typing Validity*
5. *Subject Reduction*

Conjecture

Let $\Gamma \vdash_{\mathsf{pCIC}} T : s$ *be a pCIC type such that* $\Gamma \vdash_{\mathsf{pCuIC}} t : T$*. Then there exists a term* t' *such that* $\Gamma \vdash_{\mathsf{pCIC}} t' : T$*.*

- The latter reduces the soundness of pCuIC to the soundness of pCIC:

$$\cdot \dashv_{\mathsf{pCuIC}} t : False \Rightarrow \exists t'. \cdot \dashv_{\mathsf{pCIC}} t' : False$$

Conjecture

pCuIC has the following properties:

1. *Church-Rosser property*
2. *Strong normalization*
3. *Context Validity*
4. *Typing Validity*
5. *Subject Reduction*

Conjecture

Let $\Gamma \vdash_{\mathsf{pCIC}} T : s$ *be a pCIC type such that* $\Gamma \vdash_{\mathsf{pCuIC}} t : T$*. Then there exists a term* t' *such that* $\Gamma \vdash_{\mathsf{pCIC}} t' : T$*.*

- The latter reduces the soundness of pCuIC to the soundness of pCIC:

$$\cdot \dashv_{\mathsf{pCuIC}} t : False \Rightarrow \exists t'.\ \cdot \dashv_{\mathsf{pCIC}} t' : False$$

- We prove this conjecture for the lesser pCuIC (lpCuIC), a fragment of pCuIC

Outline

- The lesser pCuIC (lpCuIC) is a fragment of pCuIC:

- The lesser pCuIC (lpCuIC) is a fragment of pCuIC:
- For any sub-derivation $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$ we have $\Gamma \vdash_{\mathsf{pCIC}} T : s$

- The lesser pCuIC (lpCuIC) is a fragment of pCuIC:
- For any sub-derivation $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$ we have $\Gamma \vdash_{\mathsf{pCIC}} T : s$
- For any sub-derivation $\Gamma \vdash_{\mathsf{lpCuIC}} T : \Pi\vec{x} : \vec{A}.s$ we have $\Gamma \vdash_{\mathsf{pCIC}} T : \Pi\vec{x} : \vec{A}.s$

- The lesser pCuIC (lpCuIC) is a fragment of pCuIC:
- For any sub-derivation $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$ we have $\Gamma \vdash_{\mathsf{pCIC}} T : s$
- For any sub-derivation $\Gamma \vdash_{\mathsf{lpCuIC}} T : \Pi \vec{x} : \vec{A}.s$ we have $\Gamma \vdash_{\mathsf{pCIC}} T : \Pi \vec{x} : \vec{A}.s$
- C-Ind is replaced by C-Ind'

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

- The lesser pCuIC (lpCuIC) is a fragment of pCuIC:
- For any sub-derivation $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$ we have $\Gamma \vdash_{\mathsf{pCIC}} T : s$
- For any sub-derivation $\Gamma \vdash_{\mathsf{lpCuIC}} T : \Pi \overrightarrow{x} : \overrightarrow{A}.s$ we have $\Gamma \vdash_{\mathsf{pCIC}} T : \Pi \overrightarrow{x} : \overrightarrow{A}.s$
- C-Ind is replaced by C-Ind'

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

- The lesser pCuIC (lpCuIC) is a fragment of pCuIC:
- For any sub-derivation $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$ we have $\Gamma \vdash_{\mathsf{pCIC}} T : s$
- For any sub-derivation $\Gamma \vdash_{\mathsf{lpCuIC}} T : \Pi \vec{x} : \vec{A}.s$ we have $\Gamma \vdash_{\mathsf{pCIC}} T : \Pi \vec{x} : \vec{A}.s$
- C-Ind is replaced by C-Ind'

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

- App is replaced by App'

$$(\text{App'})\ \frac{\Gamma \vdash t : (\Pi x : A.\ B) \quad \Gamma \vdash t' : A}{\Gamma \vdash (t\ t') : B[t'/x]} \quad (\Gamma \vdash_{\mathsf{pCIC}} t' : A \text{ or } x \notin FV(B))$$

- The lesser pCuIC (lpCuIC) is a fragment of pCuIC:
- For any sub-derivation $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$ we have $\Gamma \vdash_{\mathsf{pCIC}} T : s$
- For any sub-derivation $\Gamma \vdash_{\mathsf{lpCuIC}} T : \Pi \vec{x} : \vec{A}.s$ we have $\Gamma \vdash_{\mathsf{pCIC}} T : \Pi \vec{x} : \vec{A}.s$
- C-IND is replaced by C-IND'

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

- APP is replaced by APP'

$$\text{(APP')}\ \frac{\Gamma \vdash t : (\Pi x : A.\ B) \quad \Gamma \vdash t' : A}{\Gamma \vdash (t\ t') : B[t'/x]} \quad (\Gamma \vdash_{\mathsf{pCIC}} t' : A \text{ or } x \notin FV(B))$$

- In lpCuIC,

$\texttt{Type}_i \preceq \texttt{Type}_k$ and $\texttt{Type}_j \preceq \texttt{Type}_l \Rightarrow$ `Category@{i j}` $\preceq$ `Category@{k l}`

- The lesser pCuIC (lpCuIC) is a fragment of pCuIC:
- For any sub-derivation $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$ we have $\Gamma \vdash_{\mathsf{pCIC}} T : s$
- For any sub-derivation $\Gamma \vdash_{\mathsf{lpCuIC}} T : \Pi\vec{x} : \vec{A}.s$ we have $\Gamma \vdash_{\mathsf{pCIC}} T : \Pi\vec{x} : \vec{A}.s$
- C-Ind is replaced by C-Ind'

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

- App is replaced by App'

$$(\text{App'})\ \frac{\Gamma \vdash t : (\Pi x : A.\ B) \quad \Gamma \vdash t' : A}{\Gamma \vdash (t\ t') : B[t'/x]} \quad (\Gamma \vdash_{\mathsf{pCIC}} t' : A \text{ or } x \notin FV(B))$$

- In lpCuIC,

$$\texttt{Type}_i \preceq \texttt{Type}_k \text{ and } \texttt{Type}_j \preceq \texttt{Type}_l \Rightarrow \texttt{Category@\{i j\}} \preceq \texttt{Category@\{k l\}}$$

Also:

$$\texttt{Type}_i \preceq \texttt{Type}_k \Rightarrow \texttt{Ens@\{i\}} \preceq \texttt{Ens@\{k\}}$$

- We prove soundness of lpCuIC:

Theorem (Inhabitants in lpCuIC)

Let t and T be terms such that $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$. Then there exists t' such that $\Gamma \vdash_{\mathsf{pCIC}} t' : T$.

- We prove soundness of lpCuIC:

Theorem (Inhabitants in lpCuIC)

Let t and T be terms such that $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$. Then there exists t' such that $\Gamma \vdash_{\mathsf{pCIC}} t' : T$.

Proof sketch.

We build lifters $\Gamma \dashv_{\mathsf{pCIC}} \Upsilon_{T \preceq_{\mathsf{lpCuIC}} T'} : T \to T'$ for $T \preceq_{\mathsf{lpCuIC}} T'$.
Each sub-term $t : T$ for which we have used Conv to derive $t : T'$ is replaced with $(\Upsilon_{T \preceq_{\mathsf{lpCuIC}} T'}\ t)$. □

- We prove soundness of lpCuIC:

Theorem (Inhabitants in lpCuIC)

Let t and T be terms such that $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$. Then there exists t' such that $\Gamma \vdash_{\mathsf{pCIC}} t' : T$.

Proof sketch.

We build lifters $\Gamma \dashv_{\mathsf{pCIC}} \Upsilon_{T \preceq_{\mathsf{lpCuIC}} T'} : T \to T'$ for $T \preceq_{\mathsf{lpCuIC}} T'$.
Each sub-term $t : T$ for which we have used Conv to derive $t : T'$ is replaced with $(\Upsilon_{T \preceq_{\mathsf{lpCuIC}} T'}\ t)$. □

Corollary (Soundness of lpCuIC)

$\cdot \vdash_{\mathsf{lpCuIC}} t : False$ implies that there exists t' such that $\cdot \vdash_{\mathsf{pCIC}} t' : False$.

- Future work:
  - Proof of conjectures about pCuIC

- Future work:
  - Proof of conjectures about pCuIC
  - Implementation

- Future work:
  - Proof of conjectures about pCuIC
  - Implementation
  - Considering parameters of inductive types:

```
Inductive List@{i} (A: Type@{i}) :=
   |Nil : List@{i} A
   |Cons : A → List@{i} A → List@{i} A
.
```

- Future work:
  - Proof of conjectures about pCuIC
  - Implementation
  - Considering parameters of inductive types:

```
Inductive List@{i} (A: Type@{i}) :=
   | Nil : List@{i} A
   | Cons : A → List@{i} A → List@{i} A
.
```

We can have `List@{i} A` $\preceq$ `List@{i'} B` when `A` $\preceq$ `B`.

- Future work:
  - Proof of conjectures about pCuIC
  - Implementation
  - Considering parameters of inductive types:

```
Inductive List@{i} (A: Type@{i}) :=
   | Nil : List@{i} A
   | Cons : A → List@{i} A → List@{i} A
.
```

We can have `List@{i} A` $\preceq$ `List@{i'} B` when `A` $\preceq$ `B`.

```
Inductive Img_inh@{i j} (F : Type@{i} → Type@{j}) (A : Type@{i}) :=
   fa : (F A) → Img_inh F A
.
```

- Future work:
  - Proof of conjectures about pCuIC
  - Implementation
  - Considering parameters of inductive types:

```
Inductive List@{i} (A: Type@{i}) :=
   | Nil : List@{i} A
   | Cons : A → List@{i} A → List@{i} A
.
```

We can have `List@{i} A` $\preceq$ `List@{i'} B` when `A` $\preceq$ `B`.

```
Inductive Img_inh@{i j} (F : Type@{i} → Type@{j}) (A : Type@{i}) :=
   fa : (F A) → Img_inh F A
.
```

Whether (`Img_inh@{i j} F A`) $\preceq$ (`Img_inh@{i' j'} F B`) when `A` $\preceq$ `B` depends on variance of `F`.

- Future work:
  - Proof of conjectures about pCuIC
  - Implementation
  - Considering parameters of inductive types:

```
Inductive List@{i} (A: Type@{i}) :=
   |Nil : List@{i} A
   |Cons : A → List@{i} A → List@{i} A
.
```

    We can have `List@{i} A` $\preceq$ `List@{i'} B` when `A` $\preceq$ `B`.

```
Inductive Img_inh@{i j} (F : Type@{i} → Type@{j}) (A : Type@{i}) :=
   fa : (F A) → Img_inh F A
.
```

    Whether (`Img_inh@{i j} F A`) $\preceq$ (`Img_inh@{i' j'} F B`) when `A` $\preceq$ `B` depends on variance of `F`.
- Conclusion:
  - Presented pCuIC

- Future work:
  - Proof of conjectures about pCuIC
  - Implementation
  - Considering parameters of inductive types:

```
Inductive List@{i} (A: Type@{i}) :=
   |Nil : List@{i} A
   |Cons : A → List@{i} A → List@{i} A
.
```

    We can have List@{i} A ⪯ List@{i'} B when A ⪯ B.

```
Inductive Img_inh@{i j} (F : Type@{i} → Type@{j}) (A : Type@{i}) :=
   fa : (F A) → Img_inh F A
.
```

    Whether (Img_inh@{i j} F A) ⪯ (Img_inh@{i' j'} F B) when A ⪯ B depends on variance of F.

- Conclusion:
  - Presented pCuIC
  - Discussed how it makes working with structures such as categories and ensembles easier

- Future work:
  - Proof of conjectures about pCuIC
  - Implementation
  - Considering parameters of inductive types:

```
Inductive List@{i} (A: Type@{i}) :=
   |Nil : List@{i} A
   |Cons : A → List@{i} A → List@{i} A
.
```

    We can have `List@{i} A` $\preceq$ `List@{i'} B` when `A` $\preceq$ `B`.

```
Inductive Img_inh@{i j} (F : Type@{i} → Type@{j}) (A : Type@{i}) :=
   fa : (F A) → Img_inh F A
.
```

    Whether (`Img_inh@{i j} F A`) $\preceq$ (`Img_inh@{i' j'} F B`) when `A` $\preceq$ `B` depends on variance of `F`.

- Conclusion:
  - Presented pCuIC
  - Discussed how it makes working with structures such as categories and ensembles easier
  - Presented lpCuIC

- Future work:
  - Proof of conjectures about pCuIC
  - Implementation
  - Considering parameters of inductive types:

```
Inductive List@{i} (A: Type@{i}) :=
  | Nil : List@{i} A
  | Cons : A → List@{i} A → List@{i} A
.
```

    We can have `List@{i} A` ⪯ `List@{i'} B` when `A` ⪯ `B`.

```
Inductive Img_inh@{i j} (F : Type@{i} → Type@{j}) (A : Type@{i}) :=
  fa : (F A) → Img_inh F A
.
```

    Whether (`Img_inh@{i j} F A`) ⪯ (`Img_inh@{i' j'} F B`) when `A` ⪯ `B` depends on variance of `F`.

- Conclusion:
  - Presented pCuIC
  - Discussed how it makes working with structures such as categories and ensembles easier
  - Presented lpCuIC
    - As an intuitive reason why we believe pCuIC is sound