First Steps Towards Cumulative Inductive Types in CIC

Amin Timany and Bart Jacobs

iMinds-DistriNet, KU Leuven
`firstname.lastname@cs.kuleuven.be`

Abstract. Having the type of all types in a type system results in paradoxes like Russel's paradox. Therefore type theories like predicative calculus of inductive constructions (pCIC) – the logic of the Coq proof assistant – have a hierarchy of types $\texttt{Type}_0$, $\texttt{Type}_1$, $\texttt{Type}_2$, ..., where $\texttt{Type}_0 : \texttt{Type}_1$, $\texttt{Type}_1 : \texttt{Type}_2$, ... In a cumulative type system, e.g., pCIC, for a term `t` such that `t`: $\texttt{Type}_i$ we also have that `t`: $\texttt{Type}_{i+1}$. The system pCIC has recently been extended to support universe polymorphism, i.e., definitions can be parametrized by universe levels. This extension does not support cumulativity for inductive types. For example, we do not have that a pair of types at levels i and j is also considered a pair of types at levels $i+1$ and $j+1$.

In this paper, we discuss our on-going research on making inductive types cumulative in the pCIC. Having inductive types be cumulative alleviates some problems that occur while working with large inductive types, e.g., the category of small categories, in pCIC.

We present the pCuIC system which adds cumulativity for inductive types to pCIC and briefly discuss some of its properties and possible extensions. We, in addition, give a justification for the introduced cumulativity relation for inductive types.

1 Introduction

The type system of the proof assistant Coq, a variant of the predicative calculus of inductive constructions (pCIC) (see [3] for details), has recently been extended to support universe polymorphism [6]. There the calculus is extended with support for universe polymorphic definitions and inductive types are treated by considering copies of them at different universe levels – so long as levels satisfy constraints imposed by the inductive type and the environment. In this system the simple definition for a category, `Class Category: Type := {O: Type; H: O →O→Type; ...}` defines a type $\texttt{Category}_{ij}$ where i is the universe level for objects and j is the universe level for homomorphisms. This allows a straightforward definition of the category of small[1] categories, `Definition Cat: Category := {|O:= Category; H:= Functor;...|}` which defines a term of type category, $\texttt{Cat}_{ijkl}$:$\texttt{Category}_{ij}$, with

[1] Here, smallness and largeness are to be understood as relative to universe levels.

object type $\texttt{Category}_{kl}$[2]. However, inductive types such as `Category` not being cumulative implies that having a term `t` such that `t`: $\texttt{Category}_{kl}$ and `t` : $\texttt{Category}_{k'l'}$ is possible if and only if $k = k'$ and $l = l'$.

This side condition, however, has undesirable consequences. First and foremost, the term `Cat` above is not the category of all small categories, rather all categories at *some particular* lower universe level. Furthermore, statements about `Cat` impose restrictions on its universe levels. That is, only those copies of `Cat` that conform to the restrictions imposed are subject to the stated fact. For instance, showing that the trivial category (a category with a single object and its identity arrow) with object type `unit`: $\texttt{Type}_0$ is the terminal object of $\texttt{Cat}_{ijkl}$, implies $k = 0$. Also, showing that $\texttt{Cat}_{ijkl}$ has exponentials (functor categories) implies $j = k = l$. The latter restriction is inconsistent with the restriction $n < m$ on `TypeCat`: $\texttt{Category}_{mn}$, the category of types and functions in Coq: `Definition Type_Cat: Category := {|O:= Type; H:= fun A B ⇒ A → B;...|}`. Note that here m is the level of *type of* `O:=Type@{k}` for some k while n is the level of type of `A → B`, i.e., `Type@{k}`, hence $n = k$. This means, a copy of `Cat` cannot both have exponentials and a copy of `TypeCat` in its objects. Furthermore, having $\texttt{Cat}_{ijkl}$ cartesian closed restricts it so that $j = k = l = 0$. For further details of using universe levels to represent smallness/largeness in category theory see [7]. There, in addition to the issues mentioned above, we shortly discuss how this representation works intuitively and as expected.

It is, furthermore, noteworthy that such issues are not particular to category theory and are rather prevalent in any case incorporating large inductive types. Take the well-known definition of sets in type theory with inductive types: `Inductive Ens: Type :=ens :` Π`(A: Type),(A → Ens) → Ens`. In this case, $\texttt{Ens}_i$: $\texttt{Type}_{i+1}$ has constructor $\texttt{ens}_i$: Π (`A`: $\texttt{Type}_i$),(`A` → $\texttt{Ens}_i$) → $\texttt{Ens}_i$. As a result, the ensemble of small ensembles, `ens Ens` (λ(`x: Ens`). `x`), can't be formed as `x` in the body of the lambda-term is at a strictly lower universe level than the result ensemble.

To solve these problems, explicit lifting functions, e.g., `Lift_Ens`: $\texttt{Ens}_i \rightarrow \texttt{Ens}_j$ with $i \leq j$, could be used. They allow formation of terms such as the ensemble of *lifted* small ensembles. However, we can't prove, or even specify, Π(`t: T`), `t = Lift_T t`. As a result, working with such lifted values and types depending on them in particular is very complicated.

The rest of this paper is structured as follows: in Section 2 we present an extension of pCIC with cumulative inductive types and discuss its properties. In Section 3, we introduce lpCuIC; a subsystem of pCuIC for which we can prove soundness by reducing it to the soundness of pCIC using lifter terms. These lifters will in addition provide an intuitive reason why the cumulativity relation introduced in pCuIC is suitable. In Section 4, we conclude with discussing possible extensions to the presented system.

[2] Subject to side constraints on universe levels, e.g., $k, l < i$.

2 pCIC with Cumulative Inductive Types (pCuIC)

In this section we present the predicative calculus of cumulative inductive constructions (pCuIC for short), an extension of the predicative calculus of inductive constructions (pCIC) which additionally supports cumulativity for inductive types. The definition of pCuIC is identical to that of pCIC, except for the cumulativity rule C-Ind of Figure 2. The rules for typing judgements of this system are presented in Figure 1. In the sequel, we use $x, y, z \ldots X, Y, Z \ldots$ to denote

$$\text{(Empty)}\ \frac{}{\cdot \vdash} \qquad \text{(Decl)}\ \frac{\Gamma \vdash T : s \quad x \notin \Gamma}{\Gamma, x : T \vdash} \qquad \text{(Type)}\ \frac{\Gamma \vdash}{\Gamma \vdash \texttt{Type}_i : \texttt{Type}_{i+1}} \qquad \text{(Prop)}\ \frac{\Gamma \vdash}{\Gamma \vdash \texttt{Prop} : \texttt{Type}_i}$$

$$\text{(Var)}\ \frac{\Gamma \vdash \quad (x : T) \in \Gamma}{\Gamma \vdash x : T} \qquad \frac{\Gamma \vdash t : (\Pi x : A.B) \quad \Gamma \vdash t' : A}{\Gamma \vdash (t\ t') : B[t'/x]}\ \text{(App)}$$

$$\frac{\Gamma \vdash A : s \quad \Gamma, x : A \vdash B : s' \quad (s, s', s'') \in R_\Pi}{\Gamma \vdash \Pi x : A.\ B : s''}\ \text{(Prod)}$$

$$\text{(Lam)}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\lambda x : A.\ t) : (\Pi x : A.\ B)} \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash B : s \quad A \preceq B}{\Gamma \vdash t : B}\ \text{(Conv)}$$

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

$$\frac{I \equiv \mathsf{Ind}(X : A)\{C_1, \ldots, C_n\} \quad \Gamma \vdash I : A \quad 1 \leq i \leq n}{\Gamma \vdash \mathsf{Constr}(i, I) : C_i[I/X]}\ \text{(Constr)}$$

$$\frac{\begin{array}{c} I \equiv \mathsf{Ind}(X : \Pi \vec{x} : \vec{A}.\ s)\{C_1, \ldots, C_n\} \quad \Gamma \vdash \vec{a} : \vec{A} \quad (s, s') \in R_\xi \quad \Gamma \vdash c : (I\ \vec{a}) \\ \Gamma \vdash Q : (\Pi \vec{x} : \vec{A}.(I\ \vec{x}) \to s') \quad (\Gamma \vdash f_i : \xi(I, Q, \mathsf{Constr}(i, I), C_i) \ \ \forall 1 \leq i \leq n) \end{array}}{\Gamma \vdash \mathsf{Elim}(c, Q)\{f_1, \ldots, f_n\} : (Q\ \vec{a})\ c}\ \text{(Elim)}$$

Fig. 1. Typing Judgements

variables, $m, n, \ldots,\ M, N, \ldots$ for terms, $i, j, \ldots$ for natural numbers and s to stand for a sort, i.e., $\texttt{Prop}$ or $\texttt{Type}_i$.

2.1 Typing Rules

Figure 1 contains typing rules of PCuIC where the conversion/cumulativity relation, $\preceq$, of rule Conv is defined in Figure 2.

The relation R_Π governs the level of products formed in the system and is given by $R_\Pi = \{(\_, \texttt{Prop}, \texttt{Prop}), (\texttt{Type}_i, \texttt{Type}_j, \texttt{Type}_{max(i,j)})\}$. In other words, $\texttt{Prop}$ is impredicative while $\texttt{Type}$ is predicative. The relation R_ξ governs formation of eliminations, $R_\xi = \{(\texttt{Prop}, \texttt{Prop}), (\texttt{Type}_i, \texttt{Type}_j), (\texttt{Type}_i, \texttt{Prop})\}$ That is, we do not allow terms that are not proofs to be constructed by case analysis on a proof. The judgement $\Gamma \vdash$ expresses validity of context Γ and judgement $\Gamma \vdash t : A$ expresses the fact that term t has type A under context Γ. In case x does not appear freely in B, we abbreviate $\Pi x : A.\ B$ as $A \to B$.

Rules Ind, Constr and Elim, respectively, concern formation of inductive types, their constructor terms and their elimination. For further reading on inductive types in calculus of constructions refer to [4,5]. Here, arity for a sort s, $Ar(s)$, types strictly positive in X, $Pos(X)$ and types of constructors, $Co(X)$, are as follows:

$$Ar(s) := \Pi \vec{x} : \vec{M}.\ s \qquad Pos(X) := \Pi \vec{x} : \vec{M}.\ X\,\vec{m}$$
$$Co(X) := X\,\vec{m} \mid Pos(X) \to Co(X) \mid \Pi \vec{x} : \vec{M}.\ Co(X)$$

provided that in $Pos(X)$ and $Co(X)$, above, X does not appear in $\vec{m}$ or $\vec{M}$. This is to ensure that X appears in constructors only strictly positively. In Figure 1, ξ is the type for eliminators defined below.

Definition 1 (Eliminator Type). *Let C be a type of constructor for X and let Q and c be two terms. Then, the type of eliminator for C, $\xi(I, Q, c, C) \equiv (\xi_X(Q, c, C))[I/X]$ is defined as follows:*

$$\begin{aligned}
\xi_X(Q, c, P \to N) &= \Pi p : P.\ (\Pi \vec{x} : \vec{M}.\ (Q\ \vec{m}\ (p\ \vec{x}))) \to \xi_X(Q, (c\ p), N) \\
&\quad \textit{for } P \equiv \Pi \vec{x} : \vec{M}.\ (X\ \vec{m}) \\
\xi_X(Q, c, \Pi x : M.\ N) &= \Pi x : M.\ \xi_X(Q, (c\ x), N) \\
\xi_X(Q, c, X\ \vec{a}) &= (Q\ \vec{a}\ c)
\end{aligned}$$

■

In this system, variable x in terms $\lambda x : A.\ B$ and $\Pi x : A.\ B$ are bound in B and variable X in $Ind(X : A)\{C_1, \ldots, C_n\}$ is bound in $C_1, \ldots, C_n$. We consider terms equal up to renaming of bound variables, α-conversion. Additionally, we assume that before any substitution, if necessary, α-conversion is performed so as to prevent any variable capture.

2.2 Reduction Rules

The computational rule corresponding to inductive types, expectedly, corresponds to induction/recursion. The elimination of a term of an inductive type should perform a case analysis on its input and apply the corresponding provided elimination for that case by recursively eliminating any argument of the constructor that is of the inductive type. This will be made more clear later. For now let us consider recursors for constructors. A recursor for a constructor, as the name suggests, takes the arguments of a constructor and performs the provided elimination by recursively eliminating sub-terms. The recursor $\mu(I, F, f, C)$ for a constructor C of an inductive type I takes two terms f and F. The term f is the term that performs elimination for constructor C while term F corresponds to recursive elimination of sub-terms.

Definition 2 (Recursor). *Let C be a type of constructor for X and F and f be terms. Then, recursor $\mu(I, F, f, C) = (\mu_X(F, f, C))[I/X]$ is defined as follows:*

$$\begin{aligned}
\mu_X(F, f, P \to N) &= \lambda p : P.\mu_X(F, (f\ p\ (\lambda \vec{x} : \vec{M}.\ (F\ \vec{m}\ (p\ \vec{x})))), N) \\
&\quad \text{for } P \equiv \Pi \vec{x} : \vec{M}.(X\ \vec{m}) \\
\mu_X(F, f, \Pi x : M.\ N) &= \lambda x : M.\ \mu_X(F, (f\ x), N) \\
\mu_X(F, f, X\ \vec{a}) &= f
\end{aligned}$$

■

We consider two computation rules for pCuIC, β, for function application, $(\lambda x : A.\ t)t' \to_\beta t[t'/x]$ and ι for elimination of inductive types,

$$\mathsf{Elim}((\mathsf{Constr}(i, I)\ \vec{m}), Q)\{f_1, \ldots, f_n\} \to_\iota (\mu(I, F_{elim}(I, Q, f_1, \ldots, f_n), f_i, C_i)\ \vec{m})$$

for $I \equiv \mathsf{Ind}(X : A)\{C_1, \ldots, C_n\}$, $A \equiv \Pi \vec{x} : \vec{A}.s$ where $F_{elim}(I, Q, f_1, \ldots, f_n) \equiv \lambda \vec{x} : \vec{A}.\ \lambda c : (I\ \vec{x}).\ \mathsf{Elim}(c, Q)\{f_1, \ldots, f_n\}$. In the sequel, we write $\simeq$ to denote definitional equality, i.e., $\alpha\beta\iota\eta$-conversion. For proofs of why eliminator types and recursors above are well-typed, refer to [4,5].

As an example of inductive types and their elimination, let us define in pCuIC the prime example of inductive types, natural numbers, $nat \equiv \mathsf{Ind}(X : \mathtt{Type}_0)\{X, X \to X\}$. Let us use $Zero \equiv \mathsf{Constr}(1, nat) : nat$ and $Succ \equiv \mathsf{Constr}(2, nat) : nat \to nat$ to refer to the zero and successor constructors of the natural numbers. We construct the eliminator for type nat as follows.

$$\begin{aligned}
\cdot \vdash \Big(&\lambda Q : (nat \to s).\lambda f_1 : (Q\ Zero). \\
&\lambda f_2 : (\Pi p : nat.\ (Q\ p) \to Q\ (Succ\ p)).\ \lambda n : nat.\ \mathsf{Elim}(n, Q)\{f_1, f_2\}\Big) : \\
&\Big(\Pi Q : (nat \to s).\ (Q\ Zero) \to (\Pi p : nat.\ (Q\ p) \to Q\ (Succ\ p)) \to \Pi n : nat.\ Q\ n\Big)
\end{aligned}$$

Which is precisely the induction (in case $s = \mathtt{Prop}$) and recursion principle for natural numbers.

As another example of an inductive type consider the *even* predicate defined inductively. $even \equiv \mathsf{Ind}(X : nat \to \mathtt{Prop})\{X\ Zero, \Pi n : nat.\ X\ n \to X\ (Succ\ (Succ\ n))\}$ This type has two constructors. The first constructor constructs a proof that $Zero$ is an even number. The second constructor, takes a natural number n and a proof that n is even and produces a proof that $(Succ\ (Succ\ n))$ is even.

2.3 Cumulativity

The relation $\preceq$ in rule CONV reflects both convertibility and cumulativity. Rules for this relation are depicted in Figure 2. Rule C-IND corresponds to cumulativity of inductive types. Intuitively, rule C-IND establishes relation $I\ \vec{m} \preceq I'\ \vec{m}$, if every arity type and constructor parameter type of I is a subtype of the

$$\text{(C-Prop)}\ \frac{}{\texttt{Prop} \preceq \texttt{Type}_i} \qquad \text{(C-Type)}\ \frac{i \leq j}{\texttt{Type}_i \preceq \texttt{Type}_j} \qquad \text{(C-Prod)}\ \frac{A \simeq A' \quad B \preceq B'}{\Pi x : A.\ B \preceq \Pi x : A'.\ B'} \qquad \text{(C-Conv)}\ \frac{A \simeq B}{A \preceq B}$$

$$\frac{A \simeq A' \quad A' \preceq B' \quad B \simeq B'}{A \preceq B}\ \text{(C-Congr)}$$

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

Fig. 2. Conversion/Cumulativity Relation

corresponding type in I'. As a condition of C-Ind we have $\forall i.\ X\ \vec{m_i} \simeq X\ \vec{m'_i}$. This means that the i^{th} constructor of I and I' if applied to the same terms must produce instances of I and I' with the same values for arities.

As an example, consider the type of categories which in pCuIC is of the form: $Category_{i,j} \equiv \mathsf{Ind}(X : \texttt{Type}_{max(i+1,j+1)})\{\Pi o : \texttt{Type}_i.\Pi h : o \to o \to \texttt{Type}_j.N\}$ for $i, j \in \mathbb{N}$ where i and j don't appear in N. Clearly, we can use C-Ind to derive $Category_{i,j} \preceq Category_{k,l}$ given that $i \leq k$ and $j \leq l$. A similar argument can show that $Ens_i \preceq Ens_j$ given that $i \leq j$. Hence, pCuIC doesn't suffer from the problems mentioned earlier regarding the category of small categories and the ensemble of small ensembles.

2.4 Properties

Although we do not provide any proof, we believe that the following two conjectures, stating properties of pCuIC and relation $\preceq$, respectively, hold and can be proven in a way akin to their counterparts in [2] or [4].

Conjecture 1. pCuIC has the following properties:

1. Church-Rosser property for $\beta\iota$-reduction *(Church-Rosser)*
2. $\beta\iota$ strong normalization *(Strong Normalization)*
3. Every derivation $\Gamma \vdash t : A$ has a sub-derivation that derives $\Gamma \vdash$ and every derivation $\Gamma, x : T, \Gamma' \vdash$ has a sub-derivation that derives $\Gamma \vdash T : s$ for some sort s *(Context-Validity)*
4. if $\Gamma \vdash t : A$, then there is a sort s such that $\Gamma \vdash A : s$ *(Typing-Validity)*
5. if $\Gamma \vdash t : A$ and $t \to^*_{\beta\iota} t'$ then $\Gamma \vdash t' : A$ *(Subject Reduction)* ■

Conjecture 2. Properties of $\preceq$:

1. $\preceq$ is a partial order relation over $\simeq$: $\frac{}{t \preceq t}$ $\quad \frac{t \preceq t' \quad t' \preceq t''}{t \preceq t''}$ $\quad \frac{t \preceq t' \quad t' \preceq t}{t \simeq t'}$
2. The relation $\preceq$ is well-founded, i.e., there is no infinite decreasing chain $A_0 \succ A_1 \succ \ldots$, where $t \prec t'$ if $t \preceq t'$ and $t \not\simeq t'$ *(Well-Founded)*

3. if $\Gamma \vdash t : A$, then there exists B such that $\Gamma \vdash t : B$ and for any C such that $\Gamma \vdash t : C$ we have $B \preceq C$ *(Principal Type)* ■

The system presented in this paper, pCuIC, has a strictly richer type system compared to pCIC. In other words, $\Gamma \vdash_{\mathsf{pCIC}} t : A$ implies $\Gamma \vdash_{\mathsf{pCuIC}} t : A$ but the converse does not hold. Consider the instance of the ensemble of small ensembles expressed in pCuIC as $EE_i \equiv (\mathsf{Constr}(1, Ens_{i+1})\ Ens_i\ (\lambda x : Ens_i .\ x))\ \cdot \vdash EE_i : Ens_{i+1}$ is derivable in pCuIC but not in pCIC. The inductive type Ens_i is defined as: $Ens_i \equiv \mathsf{Ind}(X : \mathtt{Type}_{i+1})\{\Pi A : \mathtt{Type}_i .\ (A \to X) \to X\}$.

In pCuIC, Π types are considered invariant in their domain type. However, we believe that results similar to those discussed in this paper hold for the case with full contravariance for the domain type of Π types. Note that points 2 and 3 of Conjecture 2 don't hold in the version of pCuIC with full contravariance. Although, we believe they do hold in the subsystem pCuIC^n, a subsystem of PCuIC in which the universe levels are squashed such that $\mathtt{Type}_n$ is the type of all types (note that $\mathtt{Type}_n$ itself has no type in pCuIC^n). This treatment is similar to that of ECC^n in the proof of quasi normalization of ECC in [2].

For systems such as pCuIC, the strong normalization and subject reduction properties (stated in Conjecture 1 for pCuIC) imply (see [2]) the soundness and decidability of type checking. However, as of writing of this paper we have not yet proven the conjectures above. Another approach to proving soundness of pCuIC is by reducing it to the soundness of pCIC. That is, the soundness of pCuIC follows from the following conjecture:

Conjecture 3. 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 $\Gamma \vdash_{\mathsf{pCIC}} t' : T$. ■

In other words, every pCIC type that is inhabited in pCuIC is also inhabited in pCIC. We can use this conjecture to prove the soundness of pCuIC as follows. Let $False \equiv \mathsf{Ind}(X : \mathtt{Prop})\{\}$ be the inductive type with no constructors. According to the Conjecture 3, if there is a term t such that $\cdot \vdash_{\mathsf{pCuIC}} t : False$ then there is a term t' such that $\cdot \vdash_{\mathsf{pCIC}} t' : False$ which implies unsoundness of pCIC which is a contradiction. The main difficulty in proving this conjecture though is the fact that there are types in pCuIC that are not valid types in pCIC. As an example consider any type that involves the term EE_i (ensemble of small ensembles) above. Lifting such a term results in a type where the dependent argument is ensemble of lifted small ensembles (and not ensemble of small ensembles). Note that such terms and types can be part of a term which has a pCIC type. The situation is particularly complicated with the functions whose domain type is a type that is not a valid pCIC type.

3 Lesser pCuIC

In this section we introduce the lesser pCuIC (lpCuIC for short) which is a subsystem of pCuIC for which we can prove soundness by reducing it to the soundness of pCIC. This furthermore gives an intuition why the cumulativity relation introduced in this paper for inductive types is suitable.

Definition 3 (lpCuIC). *The system lpCuIC is the system pCuIC where rules* C-Ind *and* App *are replace respectively by:*

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

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

We furthermore impose the restrictions that in any derivation of $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$*, for any sub-derivation of* $\Gamma' \vdash_{\mathsf{lpCuIC}} t' : T'$ *we have* $\Gamma' \vdash_{\mathsf{pCIC}} T' : s$ *and for any sub-derivation of* $\Gamma' \vdash_{\mathsf{lpCuIC}} T' : \Pi\vec{x} : \vec{A}.s$ *we have* $\Gamma' \vdash_{\mathsf{pCIC}} T' : \Pi\vec{x} : \vec{A}.s$ ■

In other words, lpCuIC is a subsystem of pCuIC in which every valid type is also a valid type in pCIC and any functions whose output type depend on their input can't be applied to terms that are not of the appropriate type in pCIC.

In lpCuIC, we define the following lifters for the cumulativity relation. These lifters are then used to show that any type inhabited in lpCuIC is also inhabited in pCIC. This will give us a soundness proof for lpCuIC.

Definition 4 (Lifters). *Let* T *and* T' *be two terms such that* $T \preceq_{\mathsf{lpCuIC}} T'$*. Then, we define the lifter* $\Upsilon_{T \preceq_{\mathsf{lpCuIC}} T'}$ *recursively on derivation of* $T \preceq_{\mathsf{lpCuIC}} T'$*. If the last rule used to derive* $T \preceq_{\mathsf{lpCuIC}} T'$ *is:*

C-Prop, C-Type or C-Conv *then* $\Upsilon_{T \preceq_{\mathsf{lpCuIC}} T'} = \lambda x : T.\ x$

C-Prod *then* $\Upsilon_{\Pi x:A.\ B \preceq_{\mathsf{lpCuIC}} \Pi x:A'.\ B'} = \lambda f : \Pi x : A.\ B.\lambda x : A'.\ \Upsilon_{B \preceq_{\mathsf{lpCuIC}} B'}\ (f\ x)$

C-Congr *then* $\Upsilon_{A \preceq_{\mathsf{lpCuIC}} B} = \Upsilon_{A' \preceq_{\mathsf{lpCuIC}} B'}$

C-Ind *then* $\Upsilon_{I\ \vec{t} \preceq_{\mathsf{lpCuIC}} I'\ \vec{t}} = \lambda x : I\ \vec{t}.\mathsf{Elim}(x, Q)\{\phi_1, \ldots, \phi_n\}$ *for:*

$$Q \equiv \lambda\vec{y} : \vec{M_A}.\ \lambda z : I\ \vec{y}.I'\ \vec{y} \qquad \phi_i = \upsilon(I, Q, \mathsf{Constr}(i, I), C_i, \mathsf{Constr}(i, I'), C'_i)$$

$$I' \equiv \mathsf{Ind}(X : \Pi(\vec{x} : \vec{M'_A}).\ s')\{C'_1, \ldots, C'_n\}$$

$$I \equiv \mathsf{Ind}(X : \Pi(\vec{x} : \vec{M_A}).\ s)\{C_1, \ldots, C_n\}$$

Here, the constructor lifter for C*,* $\upsilon(I, Q, c, C, f, C') = \upsilon_X(Q, c, C, f, C')[I/X]$ *is defined as follows:*

$$\begin{aligned} \upsilon_X(Q, c, P \to N, f, P' \to N') = {} & \lambda p : P.\ \lambda z : (\Pi\vec{x} : \vec{M}.Q\ \vec{t}\ (p\ \vec{x})). \\ & \upsilon_X(Q, (c\ p), N, (f\ z), N') \\ & \mathit{for}\ P \equiv \Pi\vec{x} : \vec{M}.\ X\ \vec{t} \end{aligned}$$

$$\upsilon_X(Q, c, \Pi x : M.\ N, f, \Pi x : M'.\ N') = \lambda x : M.\ \upsilon_X(Q, (c\ x), N, (f\ x), N')$$
$$\upsilon_X(Q, c, X\ \vec{t}, f, X\ \vec{t}) = f$$

■

Lemma 1 (Type Correctness of Lifters). *Let T and T' be two terms such that $T \preceq_{\mathsf{lpCuIC}} T'$ and $\Gamma \vdash_{\mathsf{lpCuIC}} T : s$ and $\Gamma \vdash_{\mathsf{lpCuIC}} T' : s'$. Then, $\Gamma \vdash_{\mathsf{pCIC}} \Upsilon_{T \preceq_{\mathsf{lpCuIC}} T'} : T \to T'$.* ■

Theorem 1 (Inhabitants in lpCuIC). *Let t and T be terms such that $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$. Then there exists t' such that $\Gamma \vdash_{\mathsf{pCIC}} t' : T$.* ■

Corollary 1 (Soundness of lpCuIC). *$\cdot \vdash_{\mathsf{lpCuIC}} t : False$ implies that there exists t' such that $\cdot \vdash_{\mathsf{pCIC}} t' : False$.* ■

For proofs of the above lemma, theorem and corollary refer to [8].

The ensemble of ensembles EE_i is a valid term in lpCuIC, i.e., $\cdot \vdash_{\mathsf{lpCuIC}} EE_i : Ens_{i+1}$. For the cumulativity relation $Ens_i \preceq_{\mathsf{lpCuIC}} Ens_{i+1}$, we have the lifter:

$$\Upsilon_{Ens_i \preceq_{\mathsf{lpCuIC}} Ens_{i+1}} = \lambda x : Ens_i.\mathsf{Elim}(x, \lambda z : Ens_i.Ens_{i+1})\{\phi\}$$

for $\phi \equiv \lambda A : \mathtt{Type}_i.\lambda p : (A \to Ens_i).\lambda z : (A \to Ens_{i+1}).\mathsf{Constr}(1, Ens_{i+1})\ A\ z$. As an example, Theorem 1 gives the following term in pCIC for EE_i:

$$\mathsf{Constr}(1, Ens_{i+1})\ Ens_i\ (\lambda x : Ens_i.\Upsilon_{Ens_i \preceq_{\mathsf{lpCuIC}} Ens_{i+1}}\ x)$$

Which is the ensemble of lifted small ensembles and a valid term in pCIC.

The whole purpose of lpCuIC is to demonstrate an intuition of the workings of pCuIC and why we believe it has the properties discussed earlier. Note that although lpCuIC can express terms like ensemble of ensembles, it does not provide us with a flexible enough working environment. As an example, the type $eq\ Ens_{i+1}\ EE_i\ EE_i$ is not a valid lpCuIC type when $eq\ T$ is the equality for type T. This is due to the fact that EE_i and hence $eq\ Ens_{i+1}\ EE_i\ EE_i$ is not a valid type in pCIC. Here the inductive equality type is defined as: $eq \equiv \lambda A : \mathtt{Type}_i.\ \lambda x : A.\ \mathsf{Ind}(X : A \to \mathtt{Prop})\{X\ x\}$

4 Discussion and Conclusion

We presented pCuIC which extends pCIC with cumulativity for inductive types and discussed issues that this treatment helps mitigate. We furthermore justified the cumulativity relation for inductive types that we introduced by showing that there is a sub-system of pCuIC, lpCuIC, in which any such cumulativity relation has a corresponding lifting in pCIC. This, in addition, allowed us to reduce soundness of lpCuIC to the soundness of pCIC.

Inductive types considered lack parameters and mutual inductive types (see [5] for details). Parameters can be considered as variables in the context while an inductive type is being defined. For instance, consider the type of equality eq defined above. There, A and x are parameters of the inductive type eq. In general, the values of parameters can influence the variance of types involving them in an inductive definition. Consider $F : \mathtt{Type}_i \to \mathtt{Type}_j \vdash \mathsf{Ind}(X : \mathtt{Type}_l)\{\Pi A : \mathtt{Type}_k.(F\ A) \to X\}$. In this case we can't determine, e.g., whether $F\ A \preceq F\ B$ for $A \preceq B$. Hence separate analysis of different instances of inductive types with different parameters can help make the cumulativity results more fine-grained. In a different approach, we could add support for variables in the context, e.g., F above, to specify variance of their result with respect to their input, if appropriate, in addition to their type.

On the other hand, mutually inductive types are restricted to only appear strictly positively in one another. Therefore, although it is subject to further research, it seems natural that the approach presented here can be straightforwardly extended to the case of mutual inductive types.

Another interesting case is when we have x: $\mathtt{Type}_i$ in an inductive type. We have not considered variance of x in our relation. Doing so will result in having, e.g., $list\ A \preceq list\ B$ for $A \preceq B$. Such cumulativity relations can be very useful in practice, lessening the need of explicit conversions.

We believe that the typical ambiguity and also elaboration and unification algorithms presented in [6] can be directly extended to this system. However, as higher order unification is undecidable in general, lifting functions can be used as hints to facilitate unification when necessary. Note that these liftings are not based on case analysis on the input anymore and are hence free of the aforementioned problems.

Acknowledgements

This work was funded by EU FP7 FET-Open project ADVENT under grant number 308830.

References

1. Coquand, T., Paulin, C.: Inductively defined types. In: COLOG-88, International Conference on Computer Logic, Tallinn, USSR, Proceedings. pp. 50–66 (December 1988)
2. Luo, Z.: An Extended Calculus of Constructions. Ph.D. thesis, University of Edinbrugh, Department of Computer Science (June 1990)
3. The Coq development team: Coq 8.2 Reference Manual. Inria (2008)
4. Paulin-Mohring, C.: Inductive definitions in the system Coq - rules and properties. In: International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, Proceedings. pp. 328–345 (March 1993)
5. Paulin-Mohring, C.: Introduction to the Calculus of Inductive Constructions (Nov 2014), `https://hal.inria.fr/hal-01094195`
6. Sozeau, M., Tabareau, N.: Universe polymorphism in Coq. In: Interactive Theorem Proving, ITP 2014, Proceedings. pp. 499–514 (July 2014)
7. Timany, A., Jacobs, B.: Category theory in Coq 8.5. CoRR abs/1505.06430 (2015), `http://arxiv.org/abs/1505.06430`, accepted for a presentation at the 7th Coq workshop, Sophia Antipolis, France on June 26, 2015
8. Timany, A., Jacobs, B.: First Steps Towards Cumulative Inductive Types in CIC: Extended Version. Tech. Rep. CW684, iMinds-Distrinet, KU Leuven (March 2015), `http://www2.cs.kuleuven.be/publicaties/rapporten/cw/CW684.abs.html`