Impredicativity, Cumulativity and Product Covariance in the Logical Framework Dedukti

Thiago Felicissimo
Université Paris-Saclay, INRIA project Deducteam, Laboratoire Méthodes Formelles, ENS Paris-Saclay, France

Théo Winterhalter
Université Paris-Saclay, INRIA project Deducteam, Laboratoire Méthodes Formelles, ENS Paris-Saclay, France

Abstract

Proof assistants such as Coq implement a type theory featuring three important features: impredicativity, cumulativity and product covariance. This combination has proven difficult to be expressed in the logical framework Dedukti, and previous attempts have failed in providing an encoding that is proven confluent, sound and conservative. In this work we solve this longstanding open problem by providing an encoding of these three features that we prove to be confluent, sound and to satisfy a restricted (but, we argue, strong enough) form of conservativity. Our proof of confluence is a contribution by itself, and combines various criteria and proof techniques from rewriting theory. Our proof of soundness also contributes a new strategy in which the result is shown in terms of an inverse translation function, fixing a common flaw made in some previous encoding attempts.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Equational logic and rewriting

Keywords and phrases Dedukti, Rewriting, Confluence, Dependent types, Cumulativity, Universes

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.21

Related Version *Full Version*: `https://hal.science/hal-04470850`

Supplementary Material *Other (Artifact with supplementary data for some proofs)*: `https://github.com/Deducteam/cc-in-dk` [19]
archived at `swh:1:dir:a3e16cdbfca697ef804c9fdac8499e773a84e8ee`

Acknowledgements The authors thank François Thiré and Yoan Géran for helpful remarks about a first draft, Gaspard Férey, Jean-Pierre Jouannaud, Frédéric Blanqui and Gilles Dowek for informative discussions around the subject of this paper, and the anonymous reviewers for their helpful comments.

1 Introduction

As the number of proof systems grow, it becomes increasingly important to understand the relationship between their logics and to which extent they can be expressed in a unified setting. The research project centered around the logical framework Dedukti [7, 16] has precisely the intent of providing such a setting. By allowing for the encoding of popular logics such as predicate logic [16], higher-order logic [32, 16], set theory [17] and pure type systems [18, 22], it provides a common framework in which proofs coming from different proof systems can be rechecked, increasing the trust in their correctness. Moreover, Dedukti can then also be used for sharing these proofs with other systems, which has already allowed for exporting results to tools like Coq [15, 44], Agda [24] and HOL [44, 29].

The correctness of the verification provided by Dedukti relies however on methatheoretic results stating that the theorems that can be proven by a Dedukti encoding are exactly the same ones of the encoded logic. In the particular case of the cumulative calculus

Table 1 Comparison with previous encodings.

	Assaf* [5]	Assaf et al. [8]	Thiré* [45]	Férey^{+} [26]	This work
Confluence	✗	✓‡	✗	✗	✓
Soundness	✗†	✗†	✗†	✓	✓
Conservativity	✗	✗	✗	✗	✓⋆

†: The translation function is ill-defined (see the discussion in Section 9).
‡: Requires matching modulo ACU. ⋆: Only in a restricted form.
∗: Also handles other cumulative type systems. +: Also supports universe polymorphism.

of constructions, a type theory combining impredicativity and cumulativity with product covariance, giving an encoding satisfying these properties has remained to this day a challenge. This issue is made especially relevant by the fact that this theory is quite popular, and is most notably implemented by the proof assistant Coq.

The current situation regarding encodings of this theory is summarised in Table 1. All encodings presented until now came with a proof of *soundness*, meaning that all facts that can be proven by the encoded logic can also be proven in the encoding. However, the proofs provided by Assaf, Assaf et al. and Thiré have turned out to be incorrect, as they rely on ill-defined translation functions – see Section 9 for a detailed explanation. The situation is even more serious regarding *conservativity*, the property dual to soundness and which ensures that the encoding cannot prove more theorems than the encoded system. Indeed, none of the previous proposals have provided a proof of this fact, which is nevertheless essential to ensure that a proof checked by Dedukti is indeed correct in the original system.

One of the challenges in proving conservativity is that all known proof methods rely on confluence – which is moreover also essential to establish subject reduction. However, the combination of impredicativity, cumulativity and product covariance has proven difficult to be expressed in a confluent way in Dedukti. Indeed, almost all previous encodings have not succeeded in proving this property. A notable exception is the impressive work of Assaf et al. [8], which however relies on matching modulo ACU (assocativity-cumutativity with unit) a form of matching that is much less efficient and harder to implement than pure syntactical matching. For instance, the addition of ACU matching to the DkCheck implementation doubled the size of the kernel [20] (see also the discussion by Blanqui [14]).

In this work we address this unsatisfying state of affairs by giving an encoding of the cumulative calculus of constructions, featuring cumulativity with product covariance, that we show to satisfy the necessary metaproperties to be used in practice.

Contrary to the previous proposals, our encoding does not require non-left-linear rewrite rules, which not only are less efficient but also make confluence proofs much harder [33]. Our proof of confluence then relies on a sophisticated combination of classical results and techniques [36, 47], and automated checkers developed by the rewriting community [30, 28, 38].

With the confluence of our encoding in hand, we proceed to show soundness. In order to fix the problem with the translation function made in previous attempts, we contribute an adaptation of the technique of Winterhalter et al. [48] and Oury [39] in which the well-typedness of the translation is stated and proved in terms of an inverse translation function. The direct translation function can then be extracted from our constructive proof of soundness.

We finish by showing that our encoding satisfies a restricted form of conservativity, namely only for so-called *object terms*. We argue that, in the encoding, these are the only terms that one writes in practice, and therefore this restricted result is sufficient.

$$\textsc{EmptyCtx}\ \frac{}{\cdot \vdash} \qquad \textsc{ExtCtx}\ \frac{\Gamma \vdash A : \mathbf{Type}}{\Gamma, x : A \vdash} \qquad \textsc{Var}\ \frac{x : A \in \Gamma \quad \Gamma \vdash}{\Gamma \vdash x : A} \qquad \textsc{Cons}\ \frac{c : A \in \Sigma \quad \Gamma \vdash}{\Gamma \vdash c : A}$$

$$\textsc{Sort}\ \frac{\Gamma \vdash}{\Gamma \vdash \mathbf{Type} : \mathbf{Kind}} \qquad \textsc{Conv}\ \frac{A \equiv B \quad \Gamma \vdash t : A \qquad \Gamma \vdash B : s}{\Gamma \vdash t : B} \qquad \textsc{Pi}\ \frac{\Gamma \vdash A : \mathbf{Type} \qquad \Gamma, x : A \vdash B : s}{\Gamma \vdash (x : A) \to B : s}$$

$$\textsc{Abs}\ \frac{\Gamma \vdash A : \mathbf{Type} \qquad \Gamma, x : A \vdash B : s \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : (x : A) \to B} \qquad \textsc{App}\ \frac{\Gamma \vdash t : (x : A) \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[u/x]}$$

Figure 1 Typing rules of Dedukti.

Outline of the paper

We start in Sections 2 and 3 by recalling the definitions of Dedukti and of the variant of the calculus of constructions we consider. We then proceed in Section 4 to present the theory used in our encoding, and in Section 5 by proving its desirable properties – in particular its confluence. We define the translation function we use in Section 6, and in Sections 7 and 8 we establish the soundness and conservativity of our encoding respectively. We finish by discussing related work in Section 9, before concluding in Section 10.

2 Dedukti

We assume an underlying set $c, d, ... \in \mathcal{C}$ of *constants*, $x, y, z... \in \mathcal{V}$ of *variables* and $\mathsf{A}, \mathsf{B}, \mathsf{t}, \mathsf{u}, ... \in \mathcal{M}$ of *metavariables* equipped with an *arity* (a natural number). The *metaterms* of Dedukti [26] are defined by the following grammar.

$$\boxed{\hat{\Lambda}_{\mathsf{dk}}} \ni t, u, A, B, ... ::= x \mid c \mid \mathbf{Type} \mid \mathbf{Kind} \mid (x : A) \to B \mid \lambda x : A.t \mid t\ u \mid \mathsf{t}\{t_1, ..., t_{\mathsf{arity}(\mathsf{t})}\}$$

A *metavariable application* is written $\mathsf{t}\{t_1, ..., t_k\}$ when $\mathsf{arity}(\mathsf{t}) = k$, or just t when $\mathsf{arity}(\mathsf{t}) = 0$. The metaterms **Type** and **Kind** are called *sorts* and referred to by the letter s. We write $(x : A) \to B$ for the *dependent function type*, and whenever x does not appear free in B we write $A \to B$ instead. We define $\mathsf{fv}(t)$ as the set of free variables of t and $\mathsf{mv}(t)$ as the set of metavariables of t. When no ambiguity can arise, we allow ourselves to also write t, u, A, B for variables. We adopt the convention of writing constants names in blue font.

A *substitution* θ is a finite set of pairs t/x or $(x_1..x_k.t)/\mathsf{t}$, where $k = \mathsf{arity}(\mathsf{t})$. We write $t[\theta]$ for the application of a substitution θ to a metaterm t. The main cases of its definition are $x[\theta] = t$ when $t/x \in \theta$, and $\mathsf{t}\{u_1, ..., u_k\}[\theta] = t[u_1[\theta]/x_1, ..., u_k[\theta]/x_k]$ when $(x_1..x_k.t)/\mathsf{t} \in \theta$ – see for instance Férey [26] for the complete definition. A *rewrite system* $\mathcal{R}$ is a set of *rewrite rules*, which are pairs of the form $t \longmapsto u$ where t is of the form $c\ t_1...t_k$ and $\mathsf{fv}(t) = \mathsf{fv}(u) = \varnothing$ and $\mathsf{mv}(u) \subseteq \mathsf{mv}(t)$ and all occurrences of metavariables in t are of the form $\mathsf{t}\{x_1, ..., x_k\}$ with $x_1...x_k$ pairwise disjoint (known as the pattern condition [37]). When convenient, a rule can be given a name α, in which case we write $t \overset{\alpha}{\longmapsto} u$.

We write $\longrightarrow_{\mathcal{R}}$ for the closure under context and substitution of $\mathcal{R}$, and $\longrightarrow_{\beta\mathcal{R}}$ for $\longrightarrow_{\beta} \cup \longrightarrow_{\mathcal{R}}$ where $\longrightarrow_{\beta}$ is the usual β-reduction. Note that all these rewrite relations are defined over untyped metaterms, and that we do not consider η-reduction or -expansion, as they behave badly in the context of rewriting. We then write $\longrightarrow^*_{\beta\mathcal{R}}$ for the reflexive-transitive closure of $\longrightarrow_{\beta\mathcal{R}}$, and $\equiv_{\beta\mathcal{R}}$ for its reflexive-symmetric-transitive closure, usually called *conversion* or *definitional equality*. Most of the time $\mathcal{R}$ is clear from the context, allowing us to write just $\longrightarrow$ for $\longrightarrow_{\beta\mathcal{R}}$ and $\equiv$ for $\equiv_{\beta\mathcal{R}}$.

$$\textsc{Sub}\ \frac{n \leqslant m}{n \subseteq m} \qquad \textsc{Eq}\ \frac{A \equiv B}{A \subseteq B} \qquad \textsc{Trans}\ \frac{A \subseteq B \qquad B \subseteq C}{A \subseteq C} \qquad \textsc{ProdCov}\ \frac{A \subseteq B}{\Pi x : C.A \subseteq \Pi x : C.B}$$

$$\textsc{EmptyCtx}\ \frac{}{\cdot \vdash_{\mathsf{cc}}} \qquad \textsc{ExtCtx}\ \frac{\Gamma \vdash_{\mathsf{cc}} \qquad \Gamma \vdash_{\mathsf{cc}} A : n}{\Gamma, x : A \vdash_{\mathsf{cc}}} \qquad (x : A) \in \Gamma\ \textsc{Var}\ \frac{\Gamma \vdash_{\mathsf{cc}}}{\Gamma \vdash_{\mathsf{cc}} x : A} \qquad \textsc{Sort}\ \frac{\Gamma \vdash_{\mathsf{cc}}}{\Gamma \vdash_{\mathsf{cc}} n : \mathfrak{A}(n)}$$

$$\textsc{Pi}\ \frac{\Gamma \vdash_{\mathsf{cc}} A : n \qquad \Gamma, x : A \vdash_{\mathsf{cc}} B : m}{\Gamma \vdash_{\mathsf{cc}} \Pi x : A.B : \mathfrak{R}(n, m)} \qquad \textsc{Lam}\ \frac{\Gamma \vdash_{\mathsf{cc}} A : n \qquad \Gamma, x : A \vdash_{\mathsf{cc}} t : B}{\Gamma \vdash_{\mathsf{cc}} \lambda x : A.t : \Pi x : A.B}$$

$$\textsc{App}\ \frac{\Gamma \vdash_{\mathsf{cc}} t : \Pi x : A.B \qquad \Gamma \vdash_{\mathsf{cc}} u : A}{\Gamma \vdash_{\mathsf{cc}} t\ u : B[u/x]} \qquad A \subseteq B\ \textsc{Conv}\ \frac{\Gamma \vdash_{\mathsf{cc}} t : A \qquad \Gamma \vdash_{\mathsf{cc}} B : n}{\Gamma \vdash_{\mathsf{cc}} t : B}$$

Figure 2 Typing rules for CC.

Metavariables are useful in order to define the notion of rewrite rules, but apart from this they will have no use for us, and in particular typing will only be defined for metaterms without metavariables. Because of this, we define the set of Dedukti *terms* Λ_{dk} as the metaterms t satisfying $\mathsf{mv}(t) = \varnothing$. Given that terms will be the main object of study, from now on we adopt the convention that the letters $t, u, A, B, \ldots$ refer to terms, unless it is explicitly said that they refer to metaterms.

A *context* Γ is a finite sequence of entries of the form $x : A$. A *signature* Σ is a (possibly infinite) sequence of entries of the form $c : A$. One central notion in Dedukti is that of *theory*, which is a pair $\mathbb{T} = (\Sigma_{\mathbb{T}}, \mathcal{R}_{\mathbb{T}})$ where $\Sigma_{\mathbb{T}}$ is a signature and all constants appearing in $\mathcal{R}_{\mathbb{T}}$ are declared in $\Sigma_{\mathbb{T}}$. Theories are used in Dedukti to define the object logics in which we work (for instance, predicate logic). Given a theory $\mathbb{T}$, the typing rules of Dedukti are given in Figure 1, where the signature Σ and the conversion relation $\equiv$ are the ones defined by the theory $\mathbb{T}$. Whenever $\mathbb{T}$ is not clear from the context, we write $\mathbb{T} \triangleright \Gamma \vdash t : A$.

A signature entry $c : A$ is valid in $\mathbb{T}$ when $\mathbb{T} \triangleright \cdot \vdash A : s$ for some sort s. A theory $\mathbb{T}$ is said to be *well typed* when each entry $c : A \in \Sigma_{\mathbb{T}}$ is valid in $(\Sigma', \mathcal{R}')$, where Σ' is the prefix of $\Sigma_{\mathbb{T}}$ preceding $c : A$, and $\mathcal{R}'$ is the restriction of $\mathcal{R}_{\mathbb{T}}$ to rules only containing constants in Σ'.

3 The Cumulative Calculus of Constructions with Product Covariance

We recall the definition of the cumulative calculus of constructions with product covariance [35, 31]. It can be seen as the underlying *cumulative type system* [34, 10] of the Coq proof assistant [42], omitting the sorts Set and SProp. Its syntax is given by the following grammar.

$$\boxed{\Lambda_{\mathsf{cc}}} \ni \qquad t, u, A, B ::= x \mid n \mid \Pi x : A.B \mid \lambda x : A.t \mid t\ u$$

Here we have made the choice of representing universes directly by a natural number n. The universe that is commonly referred to as Prop then corresponds to 0, whereas Type_n corresponds to $n + 1$, allowing us to manipulate them in a more uniform way. The typing rules are then given in Figure 2, and are parametrized by the following *axiom* and *rule* functions, as they are known in the *pure type system* literature [27, Definition 4.3.2].

$$\mathfrak{A} : \mathbb{N} \to \mathbb{N} \qquad \mathfrak{R} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$
$$\mathfrak{A}(0) := 2 \qquad \mathfrak{R}(n, 0) := 0$$
$$\mathfrak{A}(1 + n) := 2 + n \qquad \mathfrak{R}(n, 1 + m) := \mathsf{max}\{n, 1 + m\}$$

▶ Remark 1. We choose to follow the implementation of Coq in placing 0 (Prop) in the universe 2 (Type_1). Some presentations choose instead to place it in 1 (Type_0) [35], a technical change that would have no impact in the strategy developed in this paper.

Compared with type systems that do not feature cumulativity, the conversion rule for CC does not only allow to exchange two types A and B when they are convertible, but also to coerce a term from type A to B when the former is a subtype of the latter. This subtyping relation, written $A \subseteq B$, is defined in the base case as $A \subseteq B$ when $A \equiv B$, or $n \subseteq m$ when $n \leqslant m$. The second rule allows us for instance to coerce a type $\Gamma \vdash A : 0$ to $\Gamma \vdash A : 1$. Then, what one calls *product covariance* is the rule allowing to deduce $\Pi x : C.A \subseteq \Pi x : C.B$ from $A \subseteq B$, which lets us for instance to coerce a function $\Gamma \vdash_{\mathsf{cc}} f : \mathsf{Nat} \to 0$ to $\Gamma \vdash_{\mathsf{cc}} f : \mathsf{Nat} \to 1$.

4 Introducing the theory $\mathbb{T}_{\mathsf{cc}}$

We now introduce the Dedukti theory $\mathbb{T}_{\mathsf{cc}}$ we will use in our encoding. We build it incrementally in order to motivate as best as possible the choices we have made.

Our first step is declaring a type $\mathfrak{S}$ along with constants 0 and S for zero and successor, allowing us to represent the CC sort n by the Dedukti term $\mathsf{S}^n\ 0$ – which from now on we write as $\underline{n}$. We then define many auxiliary constants that will be useful later, such as addition $+$, truncated predecessor P, and also constants $\mathfrak{A}$ and $\mathfrak{R}$ to represent the functions $\mathfrak{A}$ and $\mathfrak{R}$ from the definition of CC. We declare the associated rewrite rules so that they have the expected computational behavior, such as $\underline{n} + \underline{m} \longrightarrow^* \underline{n + m}$, $\underline{n} \vee \underline{m} \longrightarrow^* \underline{\mathsf{max}\{n, m\}}$, etc.

$\mathfrak{S} : \mathbf{Type}$ $\quad \mathfrak{A} : \mathfrak{S} \to \mathfrak{S}$ $\quad \mathsf{P} : \mathfrak{S} \to \mathfrak{S}$ $\quad - : \mathfrak{S} \to \mathfrak{S} \to \mathfrak{S}$ (infix)

$0 : \mathfrak{S}$ $\quad \mathfrak{A}\ 0 \longmapsto \mathsf{S}\ (\mathsf{S}\ 0)$ $\quad \mathsf{P}\ 0 \longmapsto 0$ $\quad \mathsf{l}_1 - 0 \longmapsto \mathsf{l}_1$

$\mathsf{S} : \mathfrak{S} \to \mathfrak{S}$ $\quad \mathfrak{A}\ (\mathsf{S}\ \mathsf{l}) \longmapsto \mathsf{S}\ (\mathsf{S}\ \mathsf{l})$ $\quad \mathsf{P}\ (\mathsf{S}\ \mathsf{l}) \longmapsto \mathsf{l}$ $\quad \mathsf{l}_1 - (\mathsf{S}\ \mathsf{l}_2) \longmapsto (\mathsf{P}\ \mathsf{l}_1) - \mathsf{l}_2$

$+ : \mathfrak{S} \to \mathfrak{S} \to \mathfrak{S}$ (infix) $\quad \vee : \mathfrak{S} \to \mathfrak{S} \to \mathfrak{S}$ (infix) $\quad \mathfrak{R} : \mathfrak{S} \to \mathfrak{S} \to \mathfrak{S}$

$0 + \mathsf{l}_2 \longmapsto \mathsf{l}_2$ $\quad 0 \vee \mathsf{l}_2 \longmapsto \mathsf{l}_2$ $\quad \mathfrak{R}\ \mathsf{l}_1\ 0 \longmapsto 0$

$\mathsf{l}_1 + 0 \longmapsto \mathsf{l}_1$ $\quad \mathsf{l}_1 \vee 0 \longmapsto \mathsf{l}_1$ $\quad \mathfrak{R}\ \mathsf{l}_1\ (\mathsf{S}\ \mathsf{l}_2) \longmapsto \mathsf{l}_1 \vee (\mathsf{S}\ \mathsf{l}_2)$

$(\mathsf{S}\ \mathsf{l}_1) + \mathsf{l}_2 \longmapsto \mathsf{S}\ (\mathsf{l}_1 + \mathsf{l}_2)$ $\quad (\mathsf{S}\ \mathsf{l}_1) \vee (\mathsf{S}\ \mathsf{l}_2) \longmapsto \mathsf{S}\ (\mathsf{l}_1 \vee \mathsf{l}_2)$

$\mathsf{l}_1 + (\mathsf{S}\ \mathsf{l}_2) \longmapsto \mathsf{S}\ (\mathsf{l}_1 + \mathsf{l}_2)$

Using $\mathfrak{S}$ we can then encode the universes of CC. This is done by declaring a constant U, such that the inhabitants of $\mathsf{U}_{\underline{n}}$ can then be thought of as codes for the types of CC in n. The decoding function El then maps each such code to the Dedukti type of its elements.

$\mathsf{U} : (l : \mathfrak{S}) \to \mathbf{Type}$ (written U_l) $\qquad \mathsf{El} : (l : \mathfrak{S}) \to \mathsf{U}_l \to \mathbf{Type}$ (written El_l)

Next we add constants to represent the codes inhabiting such universes. Because in CC each universe n inhabits $\mathfrak{A}(n)$, we add a constant u mapping each $l : \mathfrak{S}$ to its code in $\mathsf{U}_{(\mathfrak{A}\ l)}$. An associated rewrite rule then ensures that u_l decodes to the type U_l as expected.

$$\mathtt{u} : (l : \mathfrak{S}) \to \mathrm{U}_{(\mathfrak{R}\ l)} \quad \text{(written } \mathrm{u}_l\text{)} \qquad\qquad \mathrm{El}_{(\_)}\ \mathrm{u}_1 \overset{\mathrm{El_u}}{\longmapsto} \mathrm{U}_1$$

A similar story happens for the function type: we add a constant π mapping a code $a : \mathrm{U}_{l_a}$ and a family of codes $b : \mathrm{El}_{l_a}\ a \to \mathrm{U}_{l_b}$ to a code in $\mathrm{U}_{(\mathfrak{R}\ l_a\ l_b)}$, so that if a represents A and b represents B, then the result represents the CC type $\Pi x : A.B$. However, for reasons that will become clear later, our constant also allows us to decompose the sorts l_a and l_b into a common factor l_0 to which we apply offsets l_1 and l_2. In order to equate different decompositions of l_a and l_b, we also add a rewrite rule which removes two successors of l_1 and l_2 and compensates it by adding one in l_0. Finally, we add a rewrite rule defining the elements of $\pi^{l_0}_{l_1,l_2}\ a\ \lambda x.b$ as the Dedukti functions from the elements of a to the ones of b.

$$\begin{aligned}
&\pi : (l_0\ l_1\ l_2 : \mathfrak{S}) \to (A : \mathrm{U}_{(l_0+l_1)}) \\
&\qquad \to (B : \mathrm{El}_{(l_0+l_1)}\ A \to \mathrm{U}_{(l_0+l_2)}) \to \mathrm{U}_{(\mathfrak{R}\ (l_0+l_1)\ (l_0+l_2))} \qquad \text{(written } \pi^{l_0}_{l_1,l_2}\text{)} \\
&\pi^{l_0}_{(\mathrm{S}\ l_1),(\mathrm{S}\ l_2)}\ \mathtt{A}\ \mathtt{B} \overset{\pi_\mathrm{S}}{\longmapsto} \pi^{(\mathrm{S}\ l_0)}_{l_1,l_2}\ \mathtt{A}\ \mathtt{B} \\
&\mathrm{El}_{(_)}\ (\pi^{l_0}_{l_1,l_2}\ \mathtt{A}\ \lambda x : \mathtt{C.B}\{x\}) \overset{\mathrm{El}_\pi}{\longmapsto} (x : \mathrm{El}_{(l_0+l_1)}\ \mathtt{A}) \to \mathrm{El}_{(l_0+l_2)}\ \mathtt{B}\{x\}
\end{aligned}$$

The theory given until this point is a representation of CC *without* cumulativity, and straightforwardly applies well-known techniques from previous Dedukti encodings [18, 16]. The interesting part is for the encoding of cumulativity. The main insight of our proposal comes from the following simple result regarding the relation $\subseteq$. In the following, given a context $\Delta = x_1 : B_1 ... x_k : B_k$, let us write $\Delta \Rightarrow A$ for the CC term $\Pi x_1 : B_1 ... x_k : B_k.A$.

▶ **Lemma 2** (Case analysis of $\subseteq$). *If $A \subseteq B$ then either $A \equiv B$ or $A \longrightarrow^* \Delta \Rightarrow n$ and $B \longrightarrow^* \Delta \Rightarrow m$ for some context Δ and natural numbers n, m with $n \leqslant m$.*

Therefore, in order to simulate CC's cumulativity it suffices to add a *lift* $\uparrow$ allowing the coercion of terms from a type $\Delta \Rightarrow n$ to $\Delta \Rightarrow n+1$. However, to be able to state the type of $\uparrow$ we first need to have an internal representation for types of the form $\Delta \Rightarrow n$ in Dedukti. We do this by first defining a type for *telescopes* whose canonical elements are either the empty telescope $\blacklozenge$, or the extension $A\ l \blacktriangleleft \lambda x.D$ of a telescope D with a code A in universe U_l. We can then define a function $\Rightarrow$ that computes a Dedukti type corresponding to $\Delta \Rightarrow n$.

$$\begin{aligned}
&\mathrm{Tele} : \mathbf{Type} && \Rightarrow : \mathrm{Tele} \to \mathfrak{S} \to \mathbf{Type} \quad \text{(infix)} \\
&\blacklozenge : \mathrm{Tele} && \blacklozenge \Rightarrow \mathtt{l}_1 \overset{\Rightarrow_\blacklozenge}{\longmapsto} \mathrm{U}_{\mathtt{l}_1} \\
&\blacktriangleleft : (l : \mathfrak{S}) \to (A : \mathrm{U}_l) \to (\mathrm{El}_l\ A \to \mathrm{Tele}) && (\mathtt{A}\ l_2 \blacktriangleleft \lambda x : _.\mathtt{D}\{x\}) \Rightarrow \mathtt{l}_1 \overset{\Rightarrow_\blacktriangleleft}{\longmapsto} (x : \mathrm{El}_{l_2}\ \mathtt{A}) \to \mathtt{D}\{x\} \Rightarrow \mathtt{l}_1 \\
&\quad \to \mathrm{Tele} \quad \text{(infix, written } {}_l\blacktriangleleft\text{)}
\end{aligned}$$

With these definitions in place we can finally give the definition of $\uparrow$.[1]

$$\uparrow : (l : \mathfrak{S}) \to (D : \mathrm{Tele}) \to (D \Rightarrow l) \to (D \Rightarrow (\mathrm{S}\ l)) \quad \text{(written } \uparrow_l\text{)}$$

Because in CC the applications of cumulativity are silent, the main challenge in the encoding is to ensure that different Dedukti representations of the same CC term are convertible. The pioneering work of Assaf [4] first identified that, in a setting without product covariance, it suffices to add the following *full reflection* equations – here and in the rest of the article we write $\uparrow^m_{\underline{n}}\ D\ t$ as a notation for $\uparrow_{\underline{m-1}}\ D\ (...(\uparrow_{\underline{n}}\ D\ t)...)$ when $n \leqslant m$.

[1] Note that our lift is *single-step*, in contrast with some previous encodings [5, 45, 26] which employed a *multi-step* lift, taking a type $A : \mathrm{U}_{l_1}$ to $\uparrow^{l_2}_{l_1} \blacklozenge\ t : \mathrm{U}_{l_2}$. The avoidance of the multi-step lift is essential in order to prevent its associated non-left-linear rules, such as $\uparrow^1_1 \mathtt{D}\ \mathtt{t} \longmapsto \mathtt{t}$.

$$\pi^{0}_{\underline{1+n},\underline{m}}\ (\uparrow_{\underline{n}} \blacklozenge a)\ (\lambda x.b) \equiv \uparrow^{\frac{\mathfrak{R}(1+n,m)}{\mathfrak{R}(n,m)}} \blacklozenge (\pi^{0}_{\underline{n},\underline{m}}\ a\ (\lambda x.b))$$
$$\pi^{0}_{\underline{n},\underline{1+m}}\ a\ (\lambda x.\uparrow_{\underline{m}} \blacklozenge b) \equiv \uparrow^{\frac{\mathfrak{R}(n,1+m)}{\mathfrak{R}(n,m)}} \blacklozenge (\pi^{0}_{\underline{n},\underline{m}}\ a\ (\lambda x.b))$$

The main difficulty in implementing these as rewrite rules is that the multistep lift $\uparrow^{\underline{m}}_{\underline{n}}$ is just a notation which computes the correct number of lifts $\uparrow$ to be inserted only for a given concrete choice of n and m. For instance, if $n > m > 0$ in the second equation then no lifts should be inserted in the right hand side, whereas if $n > m = 0$ then we must insert $n-1$ lifts. Of course, this could be solved by adding these as infinite schemes of rewrite rules, for all $n, m \in \mathbb{N}$, however we want to keep the rewrite system finitary, so it can actually be used in practice in the implementation.

If only we could have more information about n and m when applying the rule, we would be able to calculate the correct amount of lifts. Thankfully, because the sorts of a and b can be decomposed with the rule π_{S}, we know that for any $\pi^{n_0}_{\underline{n_1},\underline{n_2}}\ a\ \lambda x.b$ in normal form we must have either $n_1 = 0$ or $n_2 = 0$. We can then proceed with a disjunction of cases, where in each situation we have enough information to apply the right number of lifts.

$$\pi^{0}_{(\mathrm{S}\ 1),0}\ (\uparrow_{\_} \blacklozenge \mathsf{A})\ \mathsf{B} \xmapsto{\uparrow^1_\pi} \pi^{0}_{1,0}\ \mathsf{A}\ \mathsf{B}$$
$$\pi^{(\mathrm{S}\ 1_1)}_{0,1_2}\ (\uparrow_{\_} \blacklozenge \mathsf{A})\ \mathsf{B} \xmapsto{\uparrow^2_\pi} \pi^{1_1}_{0,(\mathrm{S}\ 1_2)}\ \mathsf{A}\ \mathsf{B}$$
$$\pi^{(\mathrm{S}\ 1_1)}_{(\mathrm{S}\ 1_2),0}\ (\uparrow_{\_} \blacklozenge \mathsf{A})\ \mathsf{B} \xmapsto{\uparrow^3_\pi} \uparrow_{(\mathrm{S}\ (1_1+1_2))} \blacklozenge (\pi^{(\mathrm{S}\ 1_1)}_{1_2,0}\ \mathsf{A}\ \mathsf{B})$$

$$\Uparrow : (l : \mathfrak{S}) \to (A : \mathrm{U}_0) \to \mathrm{U}_l \quad \text{(written } \Uparrow_l\text{)}$$
$$\Uparrow_0\ \mathsf{A} \longmapsto \mathsf{A}$$
$$\Uparrow_{(\mathrm{S}\ 1)}\ \mathsf{A} \longmapsto \uparrow_1 \blacklozenge (\Uparrow_1\ \mathsf{A})$$

$$\pi^{(\mathrm{S}\ (\mathrm{S}\ 1_1))}_{1_2,0}\ \mathsf{A}\ (\lambda x : \mathsf{C}.\uparrow_{\_} \blacklozenge \mathsf{B}\{x\}) \xmapsto{\uparrow^4_\pi} \pi^{(\mathrm{S}\ 1_1)}_{(\mathrm{S}\ 1_2),0}\ \mathsf{A}\ (\lambda x : \mathsf{C}.\mathsf{B}\{x\})$$
$$\pi^{1_1}_{0,(\mathrm{S}\ 1_2)}\ \mathsf{A}\ (\lambda x : \mathsf{C}.\uparrow_{\_} \blacklozenge \mathsf{B}\{x\}) \xmapsto{\uparrow^5_\pi} \uparrow_{(1_1+1_2)} \blacklozenge (\pi^{1_1}_{0,1_2}\ \mathsf{A}\ (\lambda x : \mathsf{C}.\mathsf{B}\{x\}))$$
$$\pi^{(\mathrm{S}\ 0)}_{1,0}\ \mathsf{A}\ (\lambda x : \mathsf{C}.\uparrow_{\_} \blacklozenge \mathsf{B}\{x\}) \xmapsto{\uparrow^6_\pi} \Uparrow_{(\mathrm{S}\ 1)}\ (\pi^{0}_{(\mathrm{S}\ 1),0}\ \mathsf{A}\ (\lambda x : \mathsf{C}.\mathsf{B}\{x\}))$$

Note that in order to state the last rule we also define an auxiliary constant $\Uparrow$ which given a sort l, lifts a type from U_0 to U_l. The following proposition then ensures that we have correctly implemented Assaf's full reflection equations.

▶ **Proposition 3** (Simulation of Assaf's full reflection rules). *We have the following conversions.*

$$\pi^{0}_{\underline{1+n},\underline{m}}\ (\uparrow_l \blacklozenge a)\ (\lambda x : C.b) \equiv \uparrow^{\frac{\mathfrak{R}(1+n,m)}{\mathfrak{R}(n,m)}} \blacklozenge (\pi^{0}_{\underline{n},\underline{m}}\ a\ (\lambda x : C.b)) \tag{1}$$
$$\pi^{0}_{\underline{n},\underline{1+m}}\ a\ (\lambda x : C.\uparrow_l \blacklozenge b) \equiv \uparrow^{\frac{\mathfrak{R}(n,1+m)}{\mathfrak{R}(n,m)}} \blacklozenge (\pi^{0}_{\underline{n},\underline{m}}\ a\ (\lambda x : C.b)) \tag{2}$$

Proof. By a disjunction of cases in which each case corresponds to one of the rules $\uparrow^i_\pi$. ◀

▶ Remark 4. We note that the rules $\uparrow^i_\pi$ are also very similar to the ones identified by Assaf et al. [8]. However they also differ in a crucial way by avoiding the use of non-left-linearity and matching modulo ACU, which are less efficient and render confluence proofs much harder.

The rules given until now would ensure the uniqueness of codes for a version of CC with "simple" cumulativity. However, in a setting with product covariance we also need to ensure that $\uparrow$ properly commutes with abstraction and application. We therefore add the following two rules, which are variants of similar equations first identified by Thiré [45] and Férey [26].

$$\uparrow_1\ (\_\ (\_) \blacktriangleleft \lambda x : \_.\mathtt{D}\{x\})\ \lambda x : \mathtt{A}.\mathtt{t}\{x\} \overset{\uparrow_\lambda}{\longmapsto} \lambda x : \mathtt{A}.\uparrow_1 \mathtt{D}\{x\}\ \mathtt{t}\{x\}$$
$$\uparrow_1\ (\_\ (\_) \blacktriangleleft \lambda x : \_.\mathtt{D}\{x\})\ \mathtt{t}\ \mathtt{u} \overset{\uparrow_@}{\longmapsto} \uparrow_1 \mathtt{D}\{\mathtt{u}\}\ (\mathtt{t}\ \mathtt{u})$$

We now have almost finished presenting the theory $\mathbb{T}_{\mathsf{cc}}$. The final step is adding the following rule explaining the relationship between the elements of $\uparrow_l \blacklozenge A$ and the ones of A, which as expected should be the same. Here we have purposely avoided the expected rule $\mathrm{El}_{(\mathtt{S}\ 1)}\ (\uparrow_{(\_)} \blacklozenge \mathtt{A}) \longmapsto \mathrm{El}_1\ \mathtt{A}$ used in some previous proposals [5, 45]. This subtle difference is essential in order to allow the critical pairs between $\uparrow^i_\pi$ and El_π to close. We add a similar rule for $\Uparrow$, but once again we annotate El with $\mathtt{l}_2 - \mathtt{l}_1$ instead of 0 in order to ensure that critical pairs all close. Finally, we need a last rule similar to $\uparrow_{\mathrm{El}}$ ensuring the uniqueness of telescope representations, which will be key when proving the injectivity of $\Rightarrow$.

$$\mathrm{El}_1\ (\uparrow_{\_} \blacklozenge \mathtt{A}) \overset{\uparrow_{\mathrm{El}}}{\longmapsto} \mathrm{El}_{(\mathtt{P}\ 1)}\ \mathtt{A} \qquad \mathrm{El}_{\mathtt{l}_2}\ (\Uparrow_{\mathtt{l}_1}\ \mathtt{A}) \overset{\Uparrow_{\mathrm{El}}}{\longmapsto} \mathrm{El}_{(\mathtt{l}_2 - \mathtt{l}_1)}\ \mathtt{A} \qquad (\uparrow_{\_} \blacklozenge \mathtt{A})_1 \blacktriangleleft \mathtt{D} \overset{\uparrow_\blacktriangleleft}{\longmapsto} \mathtt{A}_{(\mathtt{P}\ 1)} \blacktriangleleft \mathtt{D}$$

5 Basic properties of $\mathbb{T}_{\mathsf{cc}}$

With the definition of the theory $\mathbb{T}_{\mathsf{cc}}$ in place, we now show that it satisfies the basic properties one expects, which will be essential for proving soundness and conservativity later. The first of them is the fact the the theory $\mathbb{T}_{\mathsf{cc}}$ is well-typed, in the sense defined in Section 2.

▶ **Proposition 5** (Well-typedness of $\mathbb{T}_{\mathsf{cc}}$). *The theory $\mathbb{T}_{\mathsf{cc}}$ is well typed.*

Proof. Checked automatically with Lambdapi – see the artifact [19] for more details. ◀

5.1 Confluence

Unlike all previous proposals, our theory $\mathbb{T}_{\mathsf{cc}}$ only makes use of left-linear rules. By preventing the use of non-left-linearity, which interacts very badly with higher-order rewriting, we have made a first step for proving confluence. Yet, confluence still does not come for free. In order to show it, we split $\beta\mathcal{R}_{\mathsf{cc}}$ into subsystems $\beta\mathcal{R}_1$ and $\mathcal{R}_2$, allowing us to apply different techniques for showing their confluence. Note that the union $\beta\mathcal{R}_1 \cup \mathcal{R}_2$ is not disjoint: the rule $\uparrow_{\mathrm{El}}$, needed for closing critical pairs in both subsystems, is shared between them.

$$\mathcal{R}_1 := \{\uparrow_@, \uparrow_\lambda, \uparrow_\blacktriangleleft, \Rightarrow_\blacklozenge, \Rightarrow_\blacktriangleleft, \uparrow_{\mathrm{El}}\} \qquad \mathcal{R}_2 := \mathcal{R}_{\mathsf{cc}} \backslash \{\uparrow_@, \uparrow_\lambda, \uparrow_\blacktriangleleft, \Rightarrow_\blacklozenge, \Rightarrow_\blacktriangleleft\}$$

Confluence of $\beta\mathcal{R}_1$

The hardest part of our proof is showing the confluence of $\beta\mathcal{R}_1$, for two main reasons. First, even though all critical pairs of $\beta\mathcal{R}_1$ close (as shown in Figure 3), because the β rule is non-normalizing on untyped terms, we cannot apply Newman's Lemma to reduce proving confluence to local confluence. Second, because the critical pairs are neither *trivial* [47] nor *development closed* [46], we cannot apply the classical criteria that avoid the use of termination. Thankfully, it turns out that we can still employ the well-known technique of showing that *orthogonal rewriting* with $\beta\mathcal{R}_1$ satisfies the diamond property, from which confluence of $\beta\mathcal{R}_1$ will follow as a simple corollary.

Given a rewrite system $\mathcal{R}$, the *orthogonal rewriting relation* $\Longrightarrow_{\beta\mathcal{R}}$ [21, 26] (also known as *developments* or *multi-step reduction* [11]) is defined over metaterms by the following inference rules, where we write $\theta \Longrightarrow \theta'$ as an abbreviation for $\mathsf{dom}(\theta) = \mathsf{dom}(\theta')$ and for all

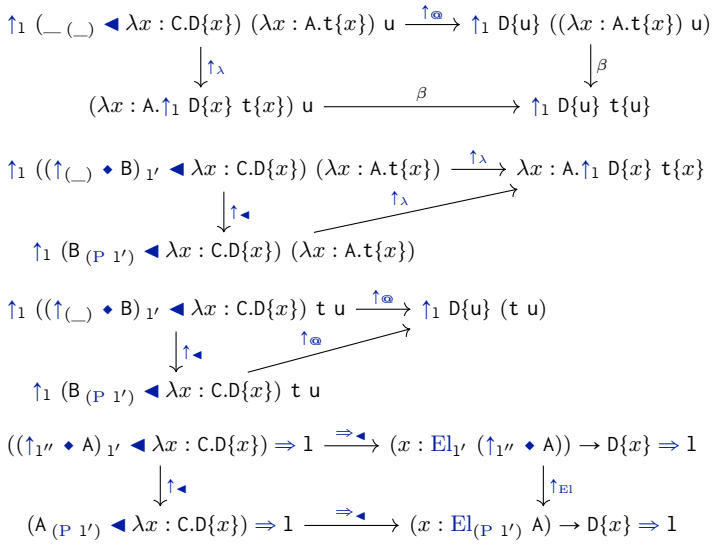

Figure 3 Critical pairs of $\beta\mathcal{R}_1$.

$\vec{x}.t/\mathtt{t} \in \theta$ and $\vec{x}.t'/\mathtt{t} \in \theta'$ we have $t \Longrightarrow t'$.

$$\textsc{Var}\ \frac{}{x \Longrightarrow x} \qquad \textsc{Const}\ \frac{}{c \Longrightarrow c} \qquad \textsc{Sort}\ \frac{}{s \Longrightarrow s} \qquad \textsc{Meta}\ \frac{t_i \Longrightarrow t'_i \quad \text{for all } i}{\mathtt{t}\{t_1..t_k\} \Longrightarrow \mathtt{t}\{t'_1..t'_k\}}$$

$$\textsc{App}\ \frac{t \Longrightarrow t' \qquad u \Longrightarrow u'}{t\ u \Longrightarrow t'\ u'} \qquad \textsc{Abs}\ \frac{A \Longrightarrow A' \qquad t \Longrightarrow t'}{\lambda x : A.t \Longrightarrow \lambda x : A'.t'} \qquad \textsc{Fun}\ \frac{A \Longrightarrow A' \qquad B \Longrightarrow B'}{(x : A) \to B \Longrightarrow (x : A') \Longrightarrow B'}$$

$$\textsc{Red}_{\mathcal{R}}\ \frac{l \longmapsto r \in \mathcal{R} \qquad \theta \Longrightarrow \theta'}{\mathsf{mv}(l) = \mathsf{dom}(\theta)\ \ l[\theta] \Longrightarrow r[\theta']} \qquad \textsc{Red}_{\beta}\ \frac{t \Longrightarrow t' \qquad u \Longrightarrow u'}{(\lambda x : A.t)\ u \Longrightarrow t'[u'/x]}$$

For all $\mathcal{R}$, orthogonal rewriting satisfies the following well-known properties – see for instance [11].

▶ **Proposition 6.** *We have* $\longrightarrow_{\beta\mathcal{R}} \subseteq \Longrightarrow_{\beta\mathcal{R}} \subseteq \longrightarrow^*_{\beta\mathcal{R}}$*, hence* $\longrightarrow^*_{\beta\mathcal{R}}$ *and* $\Longrightarrow^*_{\beta\mathcal{R}}$ *are equal.*

▶ **Proposition 7.** *If* $t \Longrightarrow_{\beta\mathcal{R}} t'$ *and* $\theta \Longrightarrow_{\beta\mathcal{R}} \theta'$ *then* $t[\theta] \Longrightarrow_{\beta\mathcal{R}} t'[\theta']$.

▶ **Proposition 8.** *We have* $t \Longrightarrow_{\beta\mathcal{R}} t$ *for all* t.

Using these properties, we can now show the following:

▶ **Proposition 9.** $\Longrightarrow_{\beta\mathcal{R}_1}$ *satisfies the diamond property.*

Proof. Given t, u, v with $u \Longleftarrow t \Longrightarrow v$ we show that there is w with $u \Longrightarrow w \Longleftarrow v$. The proof is by induction on $t \Longrightarrow u$ and $t \Longrightarrow v$. The only interesting case is when $t \Longrightarrow u$ (or dually, $t \Longrightarrow v$) is derived with rules $\textsc{Red}_{\mathcal{R}}$ or $\textsc{Red}_{\beta}$. The case $\textsc{Red}_{\beta}$ follows by the same argument as in the proof of confluence for the λ-calculus [9, Lemma 3.2.6], so let us now consider the case $\textsc{Red}_{\mathcal{R}}$, in which we have $t = l[\theta]$ for some $l \longmapsto r \in \mathcal{R}_1$ and $u = r[\theta']$ with $\theta \Longrightarrow \theta'$. There are then three possibilities regarding $t \Longrightarrow v$.

- If all applications of $\text{RED}_{\mathcal{R}}$ or RED_{β} in $t \Longrightarrow v$ occur inside the substitution θ, then because l is linear we have $v = l[\theta'']$ with $\theta \Longrightarrow \theta''$. By i.h. we have $\theta' \Longrightarrow \theta''' \Longleftarrow \theta''$ for some θ''', and thus $u = r[\theta'] \Longrightarrow r[\theta'''] \Longleftarrow l[\theta''] = v$ using Propositions 7 and 8.
- If $t \Longrightarrow v$ starts with an application of $\text{RED}_{\mathcal{R}}$ using the same rule as the one applied in $t \Longrightarrow u$, then we have $v = r[\theta'']$ with $\theta \Longrightarrow \theta''$. By i.h. we have $\theta' \Longrightarrow \theta''' \Longleftarrow \theta''$ for some θ''', and thus $u = r[\theta'] \Longrightarrow r[\theta'''] \Longleftarrow r[\theta''] = v$ using Propositions 7 and 8.
- If in $t \Longrightarrow v$ there is at least one application of $\text{RED}_{\mathcal{R}}$ with a rule $l' \longmapsto r'$ destroying the redex $t = l[\theta]$ (note that this is impossible with RED_{β}), we consider all such possible cases. In our case, this turns out to correspond exactly to the critical pairs[2] in Figure 3. We can then conclude each of these cases by appealing to the i.h. and Proposition 7 to close the diagrams. The following diagram illustrates this for when $t \Longrightarrow u$ is derived with $\uparrow_{@}$ and $t \Longrightarrow v$ destroys its redex with $\uparrow_{\lambda}$. In the middle square, X stands for C, D, A, t, u.

$$\begin{array}{ccc}
\uparrow_l\ (_\ (_) \blacktriangleleft \lambda x : C.D)\ (\lambda x : A.t)\ u & \Longrightarrow & \uparrow_{l'}\ D'[u'/x]\ ((\lambda x : A'.t')\ u') \\
\Big\Vert & & \vdots \\
 & \begin{array}{ccc} X & \Longrightarrow & X' \\ \Downarrow & i.h. & \vdots \\ X'' & \Rightarrow & X''' \end{array} & \\
(\lambda x : A''.\uparrow_{l''} D''\ t'')\ u'' & \Rightarrow & \uparrow_{l'''} D'''[u'''/x]\ t'''[u'''/x] \blacktriangleleft
\end{array}$$

Combining Proposition 9 with Proposition 6, we immediately get the following corollary.

▶ **Corollary 10.** *$\beta\mathcal{R}_1$ is confluent.*

▶ Remark 11. Alternatively, one can show the confluence of $\beta\mathcal{R}_1$ by applying a recent criterion by Dowek, Férey, Jouannaud and Liu [21, Theorem 38]. However, the proof we give is more elementary as it relies neither on orthogonal critical pairs nor on decreasing diagrams, and therefore we believe that it is accessible to a wider audience.

Confluence of $\mathcal{R}_2$

We now move to the proof of confluence of $\mathcal{R}_2$, which relies on termination.

▶ **Lemma 12.** *$\mathcal{R}_2$ is strongly normalizing.*

Proof. We translate from $\mathcal{R}_2$ into the first-order rewrite system $\hat{\mathcal{R}}_2$ obtained by forgetting about binders: $\lambda x : A.t$ is translated into $\hat{\lambda}\ A'\ t'$ and $\Pi x : A.B$ is translated into $\hat{\Pi}\ A'\ B'$, where A', B', t' are the translations of A, B, t. For instance, the rule $\uparrow^4_{\pi}$ is translated into the rule $\pi^{(\text{S}\ (\text{S}\ 1_1))}_{1_2,0}\ \mathsf{A}\ (\hat{\lambda}\ \mathsf{C}\ (\uparrow\_\ \blacklozenge\ \mathsf{B})) \longmapsto \pi^{(\text{S}\ 1_1)}_{(\text{S}\ 1_2),0}\ \mathsf{A}\ (\hat{\lambda}\ \mathsf{C}\ \mathsf{B})$. We can easily show that this interpretation preserves reduction sequences, therefore we reduce SN of $\mathcal{R}_2$ to the one of $\hat{\mathcal{R}}_2$. The latter can be shown with the use of the first-order termination checker AProVE [1, 28], and the proof can be verified by the formally certified tool CeTA [2, 43] – see the artifact [19]. ◀

▶ **Proposition 13.** *$\mathcal{R}_2$ is confluent.*

Proof. We use the tools CSI$^{\text{ho}}$ [3, 38] and SOL [30] to verify that all critical pairs of $\mathcal{R}_2$ are joinable – see the artifact [19] for details – so by Mayr and Nipkow's critical pair criterion [36, Theorem 4.7] we conclude that $\mathcal{R}_2$ is locally confluent.[3] Together with Lemma 12, this gives the confluence of $\mathcal{R}_2$ by applying Newman's Lemma. ◀

[2] Although, for arbitrary $\mathcal{R}$, it is not true in general that all such situations arise from simple critical pairs, and one needs instead to consider the more general notion of *orthogonal* critical pairs [21].

[3] Note that, although Mayr and Nipkow's criterion was shown for the specific rewrite formalism of *Higher-order Rewrite Systems (HRSs)*, following Saillard [40, Definition 5.2] we can encode the formalism of Dedukti as a specific HRS, allowing us to use their result in our setting. Alternatively, we refer to Férey's PhD thesis [26], which revisits classic confluence criteria in the rewrite formalism of Dedukti.

Confluence of $\beta\mathcal{R}_{\mathsf{cc}}$

Putting everything together, we obtain the confluence of $\beta\mathcal{R}_{\mathsf{cc}}$.

▶ **Theorem 14.** *$\beta\mathcal{R}_{\mathsf{cc}}$ is confluent.*

Proof. By Corollary 10 and Proposition 13 we have the confluence of $\beta\mathcal{R}_1$ and $\mathcal{R}_2$, and moreover the rewrite systems are left-linear and there are no critical pairs between them. Therefore, we conclude the confluence of their union by applying van Oostrom and van Raamsdonk's orthogonal combinations criterion [47, Theorem 3.13].[4] ◀

We obtain the following useful corollary, which we implicitly use in the rest of the article.

▶ **Corollary 15** (Injectivity of undefined symbols)**.** *If c is a constant that does not appear in the head of a rewrite rule, then $c\ t_1 ... t_k \equiv c\ u_1 ... u_k$ implies $t_i \equiv u_i$ for $i = 1..k$.*

5.2 Subject reduction

We start with subject reduction for β. Because we have already shown confluence of $\beta\mathcal{R}_{\mathsf{cc}}$, we obtain directly the injectivity of function types: if $(x : A) \to B \equiv (x : A') \to B'$ then $A \equiv A'$ and $B \equiv B'$. This is sufficient in order to ensure that β satisfies subject reduction.

▶ **Proposition 16** (SR_β)**.** *If $\Gamma \vdash t : A$ and $t \longrightarrow_\beta t'$ then $\Gamma \vdash t' : A$.*

Proof. Follows from the injectivity of function types [12, Lemma 31]. ◀

Moving to subject reduction for $\mathcal{R}_{\mathsf{cc}}$, the first point we realize is that this property does not hold unconditionally. For instance, the rule

$$\pi^{(\mathsf{S}\ \mathtt{l}_1)}_{0,\mathtt{l}_2}\ (\uparrow_{\_} \blacklozenge \mathsf{A})\ \mathsf{B} \longmapsto \pi^{\mathtt{l}_1}_{0,(\mathsf{S}\ \mathtt{l}_2)}\ \mathsf{A}\ \mathsf{B}$$

only preserves typing if $\mathsf{S}\ (\mathtt{l}_1[\theta] \vee (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta])) \equiv \mathtt{l}_1[\theta] \vee \mathsf{S}\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta])$, yet both sides are already in normal form. We could try to make the two sides convertible by adding a rewrite rule, however this rule would not be left-linear and thus make proving confluence much harder. Nevertheless, the fact that these terms are not convertible is actually not a problem because whenever $\mathtt{l}_1$ and $\mathtt{l}_2$ are substituted by terms of the form $\underline{n}$ for some $n \in \mathbb{N}$ then we see that the equation holds. Starting from this insight, we now show that subject reduction holds in a restricted form, which turns out to be sufficient for our needs.

We say that a term is *guarded* when all occurrences of $\Uparrow$ are of the form $\Uparrow_{\underline{n}}$ and all occurrences of π are of the form $\pi^{n_0}_{\underline{n_1},\underline{n_2}}$ for some $n, n_0, n_1, n_2 \in \mathbb{N}$. The set of guarded terms satisfies the following basic stability properties.

▶ **Proposition 17** (Stability of guarded terms under substitution and reduction)**.**

1. *If t, u are guarded then $t[u/x]$ is guarded.*
2. *If t is guarded and $t \longrightarrow t'$ then t' is guarded.*

We can now show that $\mathcal{R}_{\mathsf{cc}}$ satisfies subject reduction for guarded terms.

▶ **Proposition 18** ($\mathsf{SR}_{\mathcal{R}_{\mathsf{cc}}}$)**.** *If t is guarded and $\Gamma \vdash t : A$ and $t \longrightarrow_{\mathcal{R}_{\mathsf{cc}}} t'$ then $\Gamma \vdash t' : A$.*

[4] The same observation as in Footnote 3 applies here.

Proof. We use Lambdapi to automatically verify that the rules preserve typing (the correctness of this verification relies on the confluence of the rewrite system [41, 13], which we have by Theorem 14). The verification succeeds for all rules $l \longmapsto r \in \mathcal{R}_{\mathsf{cc}}$, except for those which do not preserve typing unconditionally. For these cases, Lambdapi reports conversion constraints on the substitution θ under which $\Gamma \vdash l[\theta] : A$ implies $\Gamma \vdash r[\theta] : A$.

1. Case $\Uparrow_{\mathrm{El}}$. Preserves typing if $\mathtt{l}_2[\theta] - \mathtt{l}_2[\theta] \equiv 0$. But by inversion of typing of the left-hand side we also get $\mathtt{l}_1[\theta] \equiv \mathtt{l}_2[\theta]$, so the rule preserves typing whenever $\mathtt{l}_1[\theta] - \mathtt{l}_1[\theta] \equiv 0$.
2. Case $\uparrow^2_\pi$. Preserves typing if $\mathrm{S}\ (\mathtt{l}_1[\theta] \vee (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta])) \equiv \mathtt{l}_1[\theta] \vee \mathrm{S}\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta])$.
3. Case $\uparrow^3_\pi$. Preserves typing if $(\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]) \vee \mathtt{l}_1[\theta] \equiv \mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]$ and $\mathrm{S}\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]) \vee \mathtt{l}_1[\theta] \equiv \mathrm{S}\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta])$.
4. Case $\uparrow^4_\pi$. Preserves typing if $\mathrm{S}\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]) \vee \mathtt{l}_1[\theta] \equiv \mathrm{S}\ ((\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]) \vee \mathtt{l}_1[\theta])$.
5. Case $\uparrow^5_\pi$. Preserves typing if $\mathtt{l}_1[\theta] \vee \mathrm{S}\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]) \equiv \mathrm{S}\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta])$ and $\mathfrak{R}\ \mathtt{l}_1[\theta]\ (\mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]) \equiv \mathtt{l}_1[\theta] + \mathtt{l}_2[\theta]$.

Because t is guarded, it follows that $\mathtt{l}_1[\theta]$ is a concrete sort in case 1, and both $\mathtt{l}_1[\theta]$ and $\mathtt{l}_2[\theta]$ are concrete sorts in the other cases, so the result follows from the fact that these equations all hold for natural numbers. ◀

▶ **Corollary 19** ($\mathsf{SR}_{\beta\mathcal{R}_{\mathsf{cc}}}$). *If t is guarded and $\Gamma \vdash t : A$ and $t \longrightarrow^* t'$ then $\Gamma \vdash t' : A$.*

▶ Remark 20. Corollary 19 guarantees that the usual type inference algorithm for Dedukti [41] is sound when Γ and t are guarded. Indeed, by inspection on its definition, if the inputs Γ and t are guarded then only guarded terms are ever reduced.

6 The translation function

Defining a Dedukti encoding usually requires specifying a *translation function* from the syntax of the source system to the one of the framework. However, whereas cumulativity is *implicit* in CC, in Dedukti it is made *explicit* by the use of a lift ($\uparrow$). Therefore, when translating a CC term, the translation function needs to figure out when to insert such lifts, even though the initial term contains no information about cumulativity. To handle this, a first idea could be to define this function only for well-typed CC terms and use typing to retrieve the missing information. However, it is not clear how to define such a function in a unique and well-founded way – see Section 9 for a detailed discussion on why.

To solve this problem, we adapt the approach of Winterhalter et al. [48] of relying instead on an *inverse translation function* $|-|$, defined from a subset of the syntax of the framework to the syntax of CC. Because the syntax of Dedukti is more explicit than the one of CC, this function can be straightforwardly defined by structural induction. Then, we can use it to state and prove soundness and conservativity. Finally, the *direct* translation function can then be recovered as the underlying algorithm of our constructive proof of soundness.

We start by carving out a subset of Dedukti's syntax over which we define $|-|$. These are the *object terms* and *object contexts*, defined by the following grammars, and where n, m ranges over natural numbers and G ranges over arbitrary guarded terms.

$$\boxed{\Lambda_o} \ni \quad t, u, A, B ::= x \mid \lambda x : \mathrm{El}_{\underline{n}}\ A.t \mid \mathrm{u}_{\underline{n}} \mid \pi^{0}_{\underline{n},\underline{m}}\ A\ \lambda x : G.B \mid \uparrow_{\underline{n}}\ G\ t \mid t\ u$$

$$\boxed{\mathsf{Ctx}_o} \ni \quad \Gamma ::= \cdot \mid \Gamma, x : \mathrm{El}_{\underline{n}}\ A$$

The *inverse translation function* can then be defined by structural induction over object terms and contexts, by the following clauses.

$$
\begin{array}{ll}
|-| : \Lambda_o \to \Lambda_{\mathsf{cc}} & \|-\| : \mathsf{Ctx}_o \to \mathsf{Ctx}_{\mathsf{cc}} \\
|x| := x & \|\cdot\| := \cdot \\
|\mathrm{u}_{\underline{n}}| := n & \|\Gamma, x : \mathrm{El}_{\underline{n}}\ A\| := \|\Gamma\|, x : |A| \\
|\lambda x : \mathrm{El}_{\underline{n}}\ A.t| := \lambda x : |A|.|t| & \\
|\pi^{0}_{\underline{n},\underline{m}}\ A\ (\lambda x : G.B)| := \Pi x : |A|.|B| & \\
|\uparrow_{\underline{n}}\ G\ t| := |t| & \\
|t\ u| := |t|\ |u| & (t\ u \text{ not of previous forms})
\end{array}
$$

Crucially, object terms are all guarded, ensuring that whenever they are well typed then their reducts also are. In addition, object terms are stable under substitution, which moreover commutes with $|-|$, two basic properties that will be essential to our proofs.

▶ **Proposition 21** (Basic properties of Λ_o and $|-|$).
1. *If $t \in \Lambda_o$ then t is guarded.*
2. *If $t, u \in \Lambda_o$ then $t[u/x] \in \Lambda_o$ and $|t|[|u|/x] = |t[u/x]|$.*

7 Soundness

Our proof of soundness requires multiple intermediate steps. We start by showing the injectivity modulo lifting of El (Proposition 23) and the injectivity of $\Rightarrow$ (Proposition 24), two technical results that are then used in the proof of *coherence* (Theorem 26), ensuring that any two different Dedukti representations of the same CC term must be convertible. With coherence in hand, we can then show that the conversion relation of CC can be reflected by the inverse translation function into the framework (Proposition 28), which then finally allow us to show the soundness of our encoding (Theorem 31).

7.1 Injectivity

We start with the following generalization of Assaf's full reflection equations, used in the proof of the injectivity of El modulo lifting. From now on, let us write $(\uparrow\_\ D)^k\ t$ for $\uparrow_{l_1}\ D\ (...(\uparrow_{l_k}\ D\ t)...)$ where the $l_1, ..., l_k$ can be any terms.

▶ **Lemma 22** (Generalized full reflection). *For all $k_1, k_2, n_1, n_2 \in \mathbb{N}$ we have*

$$\pi^{0}_{\underline{k_1+n_1},\underline{k_2+n_2}}\ ((\uparrow\_ \bullet)^{k_1}\ A)\ (\lambda x : C.(\uparrow\_ \bullet)^{k_2}\ B) \equiv \uparrow^{\mathfrak{R}(n_1+k_1,n_2+k_2)}_{\mathfrak{R}(n_1,n_2)} \bullet\ (\pi^{0}_{\underline{n_1},\underline{n_2}}\ A\ (\lambda x : C.B))$$

Proof. By induction on $k_1 + k_2$, using Proposition 3. ◀

In the following, we use the greek letter ρ to refer to rewrite sequences $t \longrightarrow^* u$. Given a rewrite sequence ρ, we write $\hbar\rho$ for the first rewrite rule applied in the head in ρ or $\hbar\rho = \bot$ if no step takes place at the head, and we write $\#\rho$ for the total number of rewrite steps in ρ. For instance, if ρ denotes the sequence

$$\mathrm{El}_l\ ((\lambda x.\uparrow_{l'} \bullet x)\ \mathrm{u}_{\underline{0}}) \longrightarrow \mathrm{El}_l\ (\uparrow_{l'} \bullet \mathrm{u}_{\underline{0}}) \longrightarrow \mathrm{El}_{(\mathrm{P}\ l)}\ \mathrm{u}_{\underline{0}} \longrightarrow \mathrm{U}_{\underline{0}}$$

then we have $\#\rho = 3$ and $\hbar\rho = \uparrow_{\mathrm{El}}$, which is the rule applied in the middle.

We can now show that the constant El is injective modulo the insertion of some lifts.

▶ **Proposition 23** (Injectivity of El modulo lifting). *If* $\mathrm{El}_{l_1}\ A_1 \equiv \mathrm{El}_{l_2}\ A_2$*, where both sides are guarded and well typed, then there are natural numbers* k_1, k_2 *such that*
(1) $A_1 \equiv (\uparrow\_\ \blacklozenge)^{k_1}\ A_0$ *and* $A_2 \equiv (\uparrow\_\ \blacklozenge)^{k_2}\ A_0$ *for some term* A_0*.*
(2) $\mathrm{S}^{k_1}\ l_0 \equiv l_1$ *and* $\mathrm{S}^{k_2}\ l_0 \equiv l_2$ *for some term* l_0*.*

Proof. Note that, under the hypotheses of the lemma, (1) implies (2), so we proceed to show that the hypotheses imply (1), however when applying the i.h. we also obtain (2) for free.

By confluence we have $\mathrm{El}_{l_1}\ A_1 \longrightarrow^* B\ {}^*\!\longleftarrow \mathrm{El}_{l_2}\ A_2$ for some B. Writing ρ_1 for $\mathrm{El}_{l_1}\ A_1 \longrightarrow^* B$ and ρ_2 for $\mathrm{El}_{l_2}\ A_2 \longrightarrow^* B$, we show the result by induction on $\#\rho_1 + \#\rho_2$, and by case analysis on $\hbar\rho_1$ and $\hbar\rho_2$. If $\hbar\rho_1$ or $\hbar\rho_2$ is $\uparrow_{\mathrm{El}}$ or $\Uparrow_{\mathrm{El}}$ then the result is easily discharged using the induction hypothesis. Otherwise, we must have $\hbar\rho_1 = \hbar\rho_2$, and the only possibilities are $\bot$ or El_{u} or El_{π}. If $\hbar\rho_1 = \hbar\rho_2 = \bot$ then the result is easily shown, and if $\hbar\rho_1 = \hbar\rho_2 = \mathrm{El}_{\mathrm{u}}$ then the result follows by using the injectivity of U (which is an undefined constant). We now illustrate the more intricate case, when $\hbar\rho_1 = \hbar\rho_2 = \mathrm{El}_{\pi}$.

For $i = 1, 2$ we can decompose ρ_i as

$$\mathrm{El}_{l_i}\ A_i \longrightarrow^* \mathrm{El}_{l'_i}\ (\pi^{m_i}_{\underline{n_i^a},\underline{n_i^b}}\ A_i^a\ \lambda x : C_i.A_i^b) \longrightarrow (x : \mathrm{El}_{(\underline{m_i + n_i^a})}\ A_i^a) \to \mathrm{El}_{(\underline{m_i + n_i^b})}\ A_i^b \overset{\rho'_i}{\longrightarrow}{}^* B$$

where the first arguments of π must be concrete sorts because these are reducts of guarded terms. In the following, we write δ for either a or b. Then it must be the case that B is of the form $(x : B^a) \to B^b$ and that we can decompose ρ'_1 and ρ'_2 into $\rho_1^a, \rho_1^b, \rho_2^a, \rho_2^b$ given by

$$\mathrm{El}_{(\underline{m_1 + n_1^\delta})}\ A_1^\delta \overset{\rho_1^\delta}{\longrightarrow}{}^* B^\delta\ {}^*\overset{\rho_2^\delta}{\longleftarrow} \mathrm{El}_{(\underline{m_2 + n_2^\delta})}\ A_2^\delta$$

We have $\#\rho_1^\delta + \#\rho_2^\delta < \#\rho_1 + \#\rho_2$, therefore by i.h. we deduce that for some terms A_0^δ, l_0^δ and natural numbers k_1^δ, k_2^δ we have (a) $A_1^\delta \equiv (\uparrow\_\ \blacklozenge)^{k_1^\delta}\ A_0^\delta$ and $A_2^\delta \equiv (\uparrow\_\ \blacklozenge)^{k_2^\delta}\ A_0^\delta$, and moreover also (b) $\underline{m_1 + n_1^\delta} \equiv \mathrm{S}^{k_1^\delta}\ l_0^\delta$ and $\underline{m_2 + n_2^\delta} \equiv \mathrm{S}^{k_2^\delta}\ l_0^\delta$.

Because $\underline{m_1 + n_1^\delta} \longrightarrow^* \underline{m_1 + n_1^\delta}$, by confluence it follows that l_0^δ also reduces to a concrete sort $p^\delta \in \mathbb{N}$. We therefore have $m_1 + n_1^\delta = k_1^\delta + p^\delta$ and $m_2 + n_2^\delta = k_2^\delta + p^\delta$. Together with the equations from (a), this allows us to show the following for $i = 1, 2$.

$$\begin{aligned} A_i \equiv \pi^{m_i}_{\underline{n_i^a},\underline{n_i^b}}\ A_i^a\ \lambda x : C_i.A_i^b &\equiv \pi^{0}_{\underline{m_i+n_i^a},\underline{m_i+n_i^b}}\ A_i^a\ \lambda x : C_i.A_i^b \\ &\equiv \pi^{0}_{\underline{k_i^a+p^a},\underline{k_i^b+p^b}}\ ((\uparrow\_\ \blacklozenge)^{k_i^a}\ A_0^a)\ (\lambda x : C_i.(\uparrow\_\ \blacklozenge)^{k_i^b}\ A_0^b) \\ &\equiv \uparrow^{\mathfrak{R}(p^a+k_i^a,p^b+k_i^b)}_{\underline{\mathfrak{R}(p^a,p^b)}} \blacklozenge\ (\pi^{0}_{\underline{p^a},\underline{p^b}}\ A_0^a\ (\lambda x : C_i.A_0^b)) \end{aligned}$$

where the last equation follows from Lemma 22. It suffices now to show that $C_1 \equiv C_2$. To see this, note that by typing constraints we must have $C_i \equiv \mathrm{El}_{\underline{m_i+n_i^a}}\ A_i^a$ and thus

$$C_i \equiv \mathrm{El}_{\underline{k_i^a+p^a}}\ ((\uparrow\_\ \blacklozenge)^{k_i^a}\ A_0^a) \equiv \mathrm{El}_{\underline{p^a}} A_0^a$$

where the right-hand side does not depend on i. ◀

The injectivity of El modulo lifting is then used to establish the injectivity of $\Rightarrow$.

▶ **Proposition 24** (Injectivity of $\Rightarrow$). *If* $D_1 \Rightarrow l_1 \equiv D_2 \Rightarrow l_2$ *and both sides are well typed and guarded, then* $D_1 \equiv D_2$ *and* $l_1 \equiv l_2$*.*

Proof. The strategy is similar to the one employed in Proposition 23. By confluence we have $D_1 \Rightarrow l_1 \longrightarrow^* B\ {}^*\!\longleftarrow D_2 \Rightarrow l_2$ for some B. By writing ρ_i for $D_i \Rightarrow l_i \longrightarrow^* B$, we show the result by induction on $\#\rho_1$ and case analysis on $\hbar\rho_1$ (which must be the same as $\hbar\rho_2$). The case $\hbar\rho_1 = \bot$ is easy, and $\hbar\rho_1 = \Rightarrow_{\blacklozenge}$ follows by injectivity of U. Finally, the case $\Rightarrow_{\blacktriangleleft}$ follows by the induction hypotheses, typing constraints and Proposition 23. ◀

7.2 Coherence

To show coherence, we first need the following technical lemma, allowing to decompose a telescope D when $D \Rightarrow l$ is convertible to a function type.

▶ **Lemma 25** (Telescope decomposition). *If* $D \Rightarrow l \equiv (x : P) \to Q$ *then* $D \longrightarrow^* A\ {}_{l'} \blacktriangleleft \lambda x : C.D'$ *for some* A, l', C, D' *with* $P \equiv \mathrm{El}_{l'}\ A$ *and* $Q \equiv D' \Rightarrow l$.

Proof. By confluence, we have $D \Rightarrow l \longrightarrow^* B \ {}^*\!\longleftarrow (x : P) \to Q$. We must have B of the form $(x : P') \to Q'$ with $P' \equiv P$ and $Q' \equiv Q$, and we can decompose $D \Rightarrow l \longrightarrow^* B$ as

$$D \Rightarrow l \longrightarrow^* (A\ {}_{l'} \blacktriangleleft \lambda x : C.D') \Rightarrow l'' \longrightarrow (x : \mathrm{El}_{l'}\ A) \to D' \Rightarrow l'' \longrightarrow^* (x : P') \to Q'$$

We thus have $D \longrightarrow^* A\ {}_{l'} \blacktriangleleft \lambda x : C.D'$ and $\mathrm{El}_{l'}\ A \equiv P' \equiv P$ and $D' \Rightarrow l \equiv Q' \equiv Q$. ◀

We now move to the proof of *coherence*, the central auxiliary result needed for soundness, ensuring that any two different Dedukti representations of the same CC term must be convertible. The actual statement of the theorem is however a bit more intricate.

▶ **Theorem 26** (Coherence). *Let* $t_1, t_2 \in \Lambda_o$ *with* $\Gamma \vdash t_1 : A_1$ *and* $\Gamma \vdash t_2 : A_2$. *If* $|t_1| = |t_2|$ *then at least one of the following holds:*

(1) $t_1 \equiv t_2$
(2) $\Gamma \vdash \uparrow^m_{\underline{n}}\ D\ t_2 : D \Rightarrow \underline{m}$ *and* $t_1 \equiv \uparrow^m_{\underline{n}}\ D\ t_2$ *for some* D *guarded*
(3) $\Gamma \vdash \uparrow^m_{\underline{n}}\ D\ t_1 : D \Rightarrow \underline{m}$ *and* $t_2 \equiv \uparrow^m_{\underline{n}}\ D\ t_1$ *for some* D *guarded*

Proof. The proof is by induction on t_1 and t_2, following the definition of $|-|$.

- Case $t_1 = \uparrow_{\underline{n}}\ D\ u$. By inversion of typing, uniqueness of type and injectivity of function types, we have $\Gamma \vdash D : \mathrm{Tele}$ and $\Gamma \vdash u : D \Rightarrow \underline{n}$. By i.h. on u and t_2, we have three cases to consider.
  - **(a)** $u \equiv t_2$. By confluence, u and t_2 have a common reduct w. Using subject reduction we know w has both types $D \Rightarrow \underline{n}$ and A_2 so by uniqueness of type, we know $D \Rightarrow \underline{n} \equiv A_2$ so we can conclude that $\Gamma \vdash t_2 : D \Rightarrow \underline{n}$ and thus that $\Gamma \vdash \uparrow_{\underline{n}}\ D\ t_2 : D \Rightarrow (\mathrm{S}\ \underline{n})$. Knowing that $t_1 \equiv \uparrow_{\underline{n}}\ D\ t_2$ by congruence, we conclude.
  - **(b)** $\Gamma \vdash \uparrow^{m'}_{\underline{n'}}\ D'\ t_2 : D' \Rightarrow \underline{m'}$ and $u \equiv \uparrow^{m'}_{\underline{n'}}\ D'\ t_2$. Similarly to above, we can show $D \Rightarrow \underline{n} \equiv D' \Rightarrow \underline{m'}$ by confluence, subject reduction and uniqueness of type. By injectivity of $\Rightarrow$ (Proposition 24) we get $D \equiv D'$ and $\underline{n} \equiv \underline{m'}$ which means $n = m'$ given that they are concrete. So $t_1 = \uparrow_{\underline{n}}\ D\ u \equiv \uparrow_{\underline{n}}\ D\ (\uparrow^n_{\underline{n'}}\ D\ t_2) = \uparrow^{1+n}_{\underline{n'}}\ D\ t_2$ by folding notations. Finally, we have $\Gamma \vdash t_2 : D' \Rightarrow \underline{n'}$, so by conversion we get $\Gamma \vdash t_2 : D \Rightarrow \underline{n'}$ and thus $\Gamma \vdash \uparrow^{1+n}_{\underline{n'}}\ D\ t_2 : D \Rightarrow \underline{1+n}$.
  - **(c)** $\Gamma \vdash \uparrow^{m'}_{\underline{n'}}\ D'\ u : D' \Rightarrow \underline{m'}$ and $t_2 \equiv \uparrow^{m'}_{\underline{n'}}\ D'\ u$. This gives us in particular that $\Gamma \vdash u : D' \Rightarrow \underline{n'}$ so by uniqueness of type we get $D \Rightarrow \underline{n} \equiv D' \Rightarrow \underline{n'}$ and thus $D \equiv D'$ and $n = n'$. If $m' = n$ then we have $t_2 \equiv u$ so we proceed as in case (a), otherwise $m' \geqslant 1 + n$ so we can conclude with $t_2 \equiv \uparrow^{m'}_{\underline{n}}\ D\ u = \uparrow^{m'}_{\underline{1+n}}\ D\ (\uparrow_{\underline{n}}\ D\ u) = \uparrow^{m'}_{\underline{1+n}}\ D\ t_1$ and $\Gamma \vdash \uparrow^{m'}_{\underline{1+n}}\ D\ t_1 : D \Rightarrow \underline{m'}$.

The case $t_2 = \uparrow_{\underline{n}}\ D\ u$ follows by the same reasoning, and for the other cases the definition of $|-|$ imposes that t_1 and t_2 must have the same head structure. Therefore, to conclude we consider t_1 and t_2 of the same form. We illustrate the following case:

- Case $t_1 = u_1\ v_1$ and $t_2 = u_2\ v_2$. By inversion we have $\Gamma \vdash u_i : (x : A_i) \to B_i$ and $\Gamma \vdash v_i : A_i$. By the i.h. applied to u_1 and u_2, we have three cases to consider:

(a) $u_1 \equiv u_2$. We thus get $A_1 \equiv A_2$ and $B_1 \equiv B_2$. Looking at the induction hypothesis on v_1 and v_2, in all cases we must have $v_1 \equiv v_2$. Indeed, if we are in cases (2) or (3) then we get $A_1 \equiv D \Rightarrow \underline{p}$ and $A_2 \equiv D \Rightarrow \underline{q}$, but together with $A_1 \equiv A_2$ this implies $p = q$, meaning that no lifts are inserted between v_1 and v_2. We thus conclude that $t_1 \equiv t_2$.

(b) $\Gamma \vdash \uparrow^m_{\underline{n}} D\ u_2 : D \Rightarrow \underline{m}$ and $u_1 \equiv \uparrow^m_{\underline{n}} D\ u_2$. Now, $(x : A_1) \to B_1 \equiv D \Rightarrow \underline{m}$, so by Lemma 25 we have $D \longrightarrow^* a\ _l \blacktriangleleft \lambda x : C.D'$ with $\mathrm{El}_l\ a \equiv A_1$ and $B_1 \equiv D' \Rightarrow \underline{m}$. Moreover, we also get that $\mathrm{El}_l\ a \equiv A_2$ and $B_2 \equiv D' \Rightarrow \underline{n}$. We are again in a situation where v_1 and v_2 share a type, so by the same arguments as in case (a) the i.h. gives $v_1 \equiv v_2$. Therefore,

$$t_1 = u_1\ v_1 \equiv (\uparrow^m_{\underline{n}} (a\ _l \blacktriangleleft \lambda x : C.D')\ u_2)\ v_2 \equiv \uparrow^m_{\underline{n}} D'[v_2/x]\ (u_2\ v_2) = \uparrow^m_{\underline{n}} D'[v_2/x]\ t_2$$

For typing, we have $\Gamma \vdash t_2 : B_2[v_2/x]$ so by conversion we have $\Gamma \vdash t_2 : D'[v_2/x] \Rightarrow \underline{n}$ and thus $\Gamma \vdash \uparrow^m_{\underline{n}} D'[v_2/x]\ t_2 : D'[v_2/x] \Rightarrow \underline{m}$.

(c) $\Gamma \vdash \uparrow^m_{\underline{n}} D\ u_1 : D \Rightarrow \underline{m}$ and $u_2 \equiv \uparrow^m_{\underline{n}} D\ u_1$. Symmetric to case (b). ◀

7.3 Reflection of conversion

With coherence in hand, we can show that the conversion relation of CC can be reflected by the inverse translation function into the framework. As an intermediate lemma, we first need to show that individual reduction steps of CC can be simulated in Dedukti.

▶ **Lemma 27** (Simulation of reduction steps). *Let $t \in \Lambda_o$ with $\Gamma \vdash t : A$ and $|t| \longrightarrow u$ for some $u \in \Lambda_{\mathsf{CC}}$. Then, there is some $t' \in \Lambda_o$ such that $|t'| = u$ and $t \longrightarrow^* t'$.*

Proof. By induction on t, following the definition of Λ_o. Almost all cases are either impossible, or follow by applying the i.h. to the subterm being reduced. The only interesting case is when $t = t_1\ t_2$ and the reduction happens in the head. Then, the only possibility is that $t_1 = \uparrow_{\underline{n_k}} D_k\ (...(\uparrow_{\underline{n_1}} D_1\ v)...)$ with $v = \lambda x : C.s$ and $|t| = (\lambda x : |C|.|s|)\ |t_2| \longrightarrow |s|[|t_2|/x]$. If $k = 0$ then the result is immediate, as t is a β redex. Otherwise, by typing constraints and Proposition 24 we can see that we have $D_1 \equiv ... \equiv D_k$ and $n_{i+1} = n_i + 1$ for $i = 1..k-1$, so by confluence we have some common reduct D_0 of all of them so that $t_1 \longrightarrow^* \uparrow^{n_k+1}_{\underline{n_1}} D_0\ v$. Then, by inversion of typing, v has both types $D_0 \Rightarrow \underline{n_1}$ and $(x : C) \to A'$ for some A', hence by uniqueness of types we have $D_0 \Rightarrow \underline{n_1} \equiv (x : C) \to A'$, which by Lemma 25 implies $D_0 \longrightarrow^* C'\ _l \blacktriangleleft \lambda x : B.D'$ for some C', l, B, D'. Abbreviating $C'\ _l \blacktriangleleft \lambda x : B.D'$ as D'_0,

$$t \longrightarrow^* \uparrow^{n_k+1}_{\underline{n_1}} D'_0\ (\lambda x : C.s)\ t_2 \longrightarrow^* (\lambda x : C.\uparrow^{n_k+1}_{\underline{n_1}} D'\ s)\ t_2 \longrightarrow \uparrow^{n_k+1}_{\underline{n_1}} D'[t_2/x]\ s[t_2/x]$$

and we have $\uparrow^{n_k+1}_{\underline{n_1}} D'[t_2/x]\ s[t_2/x] \in \Lambda_o$, with $|\uparrow^{n_k+1}_{\underline{n_1}} D'[t_2/x]\ s[t_2/x]| = |s|[|t_2|/x]$. ◀

▶ **Proposition 28** (Reflection of type conversion). *Let $A, B \in \Lambda_o$ with $\Gamma \vdash A : \mathrm{U}_{\underline{n}}$ and $\Gamma \vdash B : \mathrm{U}_{\underline{m}}$. If $|A| \equiv |B|$ then $\mathrm{El}_{\underline{n}}\ A \equiv \mathrm{El}_{\underline{m}}\ B$.*

Proof. Take $k := \max\{n, m\}$; we have $\Gamma \vdash \uparrow^k_{\underline{n}} \blacklozenge A : \mathrm{U}_{\underline{k}}$ and $\Gamma \vdash \uparrow^k_{\underline{m}} \blacklozenge B : \mathrm{U}_{\underline{k}}$ and $|\uparrow^k_{\underline{n}} \blacklozenge A| = |A| \equiv |B| = |\uparrow^k_{\underline{m}} \blacklozenge B|$. By confluence we have $|\uparrow^k_{\underline{n}} \blacklozenge A| \longrightarrow^* C\ {}^*\!\longleftarrow |\uparrow^k_{\underline{m}} \blacklozenge B|$ for some C. By iterating Lemma 27 with subject reduction, we get $\uparrow^k_{\underline{n}} \blacklozenge A \longrightarrow^* A'$ and $\uparrow^k_{\underline{m}} \blacklozenge B \longrightarrow^* B'$ and $|A'| = C = |B'|$ for some A' and B'. We also have $\Gamma \vdash A' : \mathrm{U}_{\underline{k}}$ and $\Gamma \vdash B' : \mathrm{U}_{\underline{k}}$, so by Theorem 26 we get $A' \equiv B'$ – note that because A' and B' have the same type, there can be no lifts between them. Therefore, we have $\uparrow^k_{\underline{n}} \blacklozenge A \equiv \uparrow^k_{\underline{m}} \blacklozenge B$ and thus we conclude $\mathrm{El}_{\underline{n}}\ A \equiv \mathrm{El}_{\underline{k}}\ (\uparrow^k_{\underline{n}} \blacklozenge A) \equiv \mathrm{El}_{\underline{k}}\ (\uparrow^k_{\underline{m}} \blacklozenge B) \equiv \mathrm{El}_{\underline{m}}\ B$. ◀

7.4 Soundness

We now have almost all auxiliary results needed for showing soundness. As a last step, we only need the following two easy lemmas.

▶ **Lemma 29** (Computing the El of a translation). *Let $A \in \Lambda_o$ with $\mathrm{El}_l\ A$ well typed.*
1. *If $|A| = n$ then $\mathrm{El}_l\ A \longrightarrow^* \mathrm{U}_{\underline{n}}$.*
2. *If $|A| = \Pi x : A_1.A_2$ then $\mathrm{El}_l\ A \longrightarrow^* (x : \mathrm{El}_{\underline{n_1}}\ A_1') \to \mathrm{El}_{\underline{n_2}}\ A_2'$ with $|A_i'| = A_i$.*

Proof. By definition of $|-|$ and typing constraints. ◀

▶ **Lemma 30** (Telescope translation). *Let $A_1, A_2 \in \Lambda_o$ with $\Gamma \vdash A_i : \mathrm{U}_{\underline{n_i}}$. If $|A_i| = \Delta \Rightarrow m_i$ for some $m_1 \leqslant m_2$, then we have $\mathrm{El}_{\underline{n_i}}\ A_i \equiv D \Rightarrow \underline{m_i}$ for some guarded D with $\Gamma \vdash D : \mathrm{Tele}$.*

Proof. By induction on Δ. ◀

▶ **Theorem 31** (Soundness). *If $\Gamma \vdash_{\mathsf{cc}} t : A$ then we have $\Gamma' \vdash t' : \mathrm{El}_{\underline{n}}\ A'$ for some $\Gamma' \in \mathsf{Ctx}_o$ and $t', A' \in \Lambda_o$ and $n \in \mathbb{N}$ with $\|\Gamma'\| = \Gamma$ and $|t'| = t$ and $|A'| = A$.*

Proof. We instead show the following two points, which together imply the theorem.
- If $\Gamma \vdash_{\mathsf{cc}}$ then $\Gamma' \vdash$ for some $\Gamma' \in \mathsf{Ctx}_o$ with $\|\Gamma'\| = \Gamma$.
- If $\Gamma \vdash_{\mathsf{cc}} t : A$ and $\Gamma' \vdash$ for some $\Gamma' \in \mathsf{Ctx}_o$ with $\|\Gamma'\| = \Gamma$ then $\Gamma' \vdash t' : \mathrm{El}_{\underline{n}}\ A'$ for some $n \in \mathbb{N}$ and $A', t' \in \Lambda_o$ with $|A'| = A$ and $|t'| = t$.

We prove them by induction on the derivation of $\Gamma \vdash_{\mathsf{cc}}$ or $\Gamma \vdash_{\mathsf{cc}} t : A$, and illustrate here two interesting cases.

- Case

$$\textsc{Lam}\ \frac{\Gamma \vdash_{\mathsf{cc}} A : n \qquad \Gamma, x : A \vdash_{\mathsf{cc}} t : B}{\Gamma \vdash_{\mathsf{cc}} \lambda x : A.t : \Pi x : A.B}$$

  By i.h. and Lemma 29 we have $\Gamma' \vdash A' : \mathrm{U}_{\underline{n}}$ and $|A'| = A$. Therefore we have $\Gamma', x : \mathrm{El}_{\underline{n}}\ A' \vdash$, so by i.h. we get $\Gamma', x : \mathrm{El}_{\underline{n}}\ A' \vdash t' : \mathrm{El}_{\underline{m}}\ B'$ for some m and with $|t'| = t$ and $|B'| = B$. By inversion, we then deduce $\Gamma', x : \mathrm{El}_{\underline{n}}\ A' \vdash B' : \mathrm{U}_{\underline{m}}$. We can now show $\Gamma' \vdash \lambda x : \mathrm{El}_{\underline{n}}\ A'.t' : (x : \mathrm{El}_{\underline{n}}\ A') \to \mathrm{El}_{\underline{m}}\ B'$ and because its type is convertible to $\mathrm{El}_{\underline{\Re(n,m)}}\ (\pi^{0}_{\underline{n},\underline{m}}\ A'\ (\lambda x : \mathrm{El}_{\underline{n}}\ A'.B'))$, which is well typed, we conclude by applying the conversion rule.
- Case

$$A \subseteq B\ \textsc{Conv}\ \frac{\Gamma \vdash_{\mathsf{cc}} t : A \qquad \Gamma \vdash_{\mathsf{cc}} B : n}{\Gamma \vdash_{\mathsf{cc}} t : B}$$

  By induction hypothesis we have $\Gamma' \vdash t' : \mathrm{El}_{\underline{m}}\ A'$ and $\Gamma' \vdash B' : \mathrm{U}_{\underline{n}}$ with $|t'| = t$, $|A'| = A$ and $|B'| = B$ (using Lemma 29 for the second derivation). By inversion we obtain $\Gamma' \vdash A' : \mathrm{U}_{\underline{m}}$. We now use Lemma 2 to split $A \subseteq B$ into two cases:
  - $A \equiv B$. We have $|A'| \equiv |B'|$ so by Proposition 28 we conclude $\mathrm{El}_{\underline{m}}\ A' \equiv \mathrm{El}_{\underline{n}}\ B'$, and thus $\Gamma' \vdash t' : \mathrm{El}_{\underline{n}}\ B'$.
  - $A \longrightarrow^* \Delta \Rightarrow p$ and $B \longrightarrow^* \Delta \Rightarrow q$ with $p \leqslant q$. We apply Lemma 27 on A' to get some A'' such that $|A''| = \Delta \Rightarrow p$ and $A' \longrightarrow^* A''$. Similarly, we get B'' with $|B''| = \Delta \Rightarrow q$ and $B' \longrightarrow^* B''$. We can then apply Lemma 30 to obtain a guarded term D such that $\Gamma' \vdash D : \mathrm{Tele}$ and $\mathrm{El}_{\underline{m}}\ A'' \equiv D \Rightarrow \underline{p}$ and $\mathrm{El}_{\underline{n}}\ B'' \equiv D \Rightarrow \underline{q}$. We can now conclude with $\Gamma' \vdash \uparrow^{\underline{q}}_{\underline{p}}\ D\ t : \mathrm{El}_{\underline{n}}\ B'$. ◀

8 Conservativity

Now that we have seen that our encoding is sound, we can move to the proof of conservativity. The usual statement of conservativity (using direct translation functions $[-] : \Lambda_{\mathsf{cc}} \to \Lambda_{\mathsf{dk}}$ and $[\![-]\!] : \mathsf{Ctx}_{\mathsf{cc}} \to \mathsf{Ctx}_{\mathsf{dk}}$) would say that, given Γ, A satisfying $\Gamma \vdash_{\mathsf{cc}} A : n$, if $[\![\Gamma]\!] \vdash t : \mathrm{El}_{\underline{n}}\ [A]$ then we have $\Gamma \vdash_{\mathsf{cc}} t' : A$ for some t'. When rephrasing this statement with the inverse translation function $|-|$, the full conservativity property would then assert that, for $\Gamma \in \mathsf{Ctx}_o$ and $A \in \Lambda_o$ with $\|\Gamma\| \vdash |A| : n$, if $\Gamma \vdash t : \mathrm{El}_{\underline{n}}\ A$ then $\|\Gamma\| \vdash_{\mathsf{cc}} t' : |A|$ for some t'.

In the following, we instead show *conservativity for object terms*, a restricted form of conservativity in which the witness t of the typing judgment $\Gamma \vdash t : \mathrm{El}_{\underline{n}}\ A$ is required to be an object term. We argue that this is enough because in practice the object terms are the only ones a user of the encoding (or an automatic translator) would write. Nevertheless, it should be possible to strengthen our result to obtain full conservativity, as discussed in the conclusion.

The first step in our proof is showing that $|-|$ preserves definitional equality. This is however not immediate, because $|-|$ does not preserve reduction steps. Fortunately, we can define an auxiliary function $|-|^\bullet$ extending $|-|$ that satisfies this property. We start by defining the *extended object terms* $\Lambda_o^\bullet$ which will be used as the domain of $|-|^\bullet$. Here we write G, G' for any guarded terms, and n, n_0, n_1, n_2 for any natural numbers.

$$\boxed{\Lambda_o^\bullet} \ni \quad t, u, A, B ::= x \mid (x : A) \to B \mid \lambda x : A.t \mid \mathrm{U}_{\underline{n}} \mid \mathrm{El}_G\ A \mid \mathrm{u}_{\underline{n}}$$
$$\mid \pi^{n_0}_{\underline{n_1},\underline{n_2}}\ A\ \lambda x : G.B \mid \uparrow_G G'\ t \mid \Uparrow_{\underline{n}} t \mid t\ u$$

The function $|-|^\bullet$ is then defined by the following clauses.

$$|-|^\bullet : \Lambda_o^\bullet \to \Lambda_{\mathsf{cc}}$$

$\lvert x\rvert^\bullet := x$	$\lvert \mathrm{El}_G\ A\rvert^\bullet := \lvert A\rvert^\bullet$	$\lvert (x : A) \to B\rvert^\bullet := \Pi x : \lvert A\rvert^\bullet . \lvert B\rvert^\bullet$
$\lvert \mathrm{u}_{\underline{n}}\rvert^\bullet := n$	$\lvert \uparrow_G G'\ t\rvert^\bullet := \lvert t\rvert^\bullet$	$\lvert \lambda x : A.t\rvert^\bullet := \lambda x : \lvert A\rvert^\bullet . \lvert t\rvert^\bullet$
$\lvert \mathrm{U}_{\underline{n}}\rvert^\bullet := n$	$\lvert \Uparrow_{\underline{n}} t\rvert^\bullet := \lvert t\rvert^\bullet$	$\lvert \pi^{n_0}_{\underline{n_1},\underline{n_2}}\ A\ (\lambda x : G.B)\rvert^\bullet := \Pi x : \lvert A\rvert^\bullet . \lvert B\rvert^\bullet$
		$\lvert t\ u\rvert^\bullet := \lvert t\rvert^\bullet\ \lvert u\rvert^\bullet$ ($t\ u$ not of previous forms)

We can show that $|-|^\bullet$ satisfies many desirable properties, among them being the preservation of reduction steps and thus also of definitional equality by $|-|^\bullet$.

▶ **Lemma 32** (Basic properties of $\Lambda_o^\bullet$ and $|-|^\bullet$).

1. *$\Lambda_o^\bullet$ is a superset of Λ_o, and $|-|^\bullet$ restricts to $|-|$ in Λ_o.*
2. *If $t \in \Lambda_o^\bullet$ then t is guarded.*
3. *If $t, u \in \Lambda_o^\bullet$ then $t[u/x] \in \Lambda_o^\bullet$ and $|t|^\bullet[|u|^\bullet/x] = |t[u/x]|^\bullet$.*
4. *If $t \in \Lambda_o^\bullet$ and $t \longrightarrow^* u$ then $u \in \Lambda_o^\bullet$ and $|t|^\bullet \longrightarrow^* |u|^\bullet$.*
5. *If $t, u \in \Lambda_o^\bullet$ and $t \equiv u$ then $|t|^\bullet \equiv |u|^\bullet$.*

Using these basic properties, we can now show conservativity.

▶ **Theorem 33** (Conservativity for object terms). *Let $\Gamma \in \mathsf{Ctx}_o$ and $A \in \Lambda_o$ with $\|\Gamma\| \vdash_{\mathsf{cc}} |A| : n$ for some n. If $\Gamma \vdash t : \mathrm{El}_{\underline{n}}\ A$ with t an object term, then we have $\|\Gamma\| \vdash_{\mathsf{cc}} |t| : |A|$.*

Proof. We instead show the following claim.

▷ Claim 34. Let $\Gamma \vdash t : A$ with $\Gamma \in \mathsf{Ctx}_o$ and $\|\Gamma\| \vdash_{\mathsf{cc}}$. If t is an object term, then there exists $A' \in \Lambda_o^\bullet$ with $A \equiv A'$ and $\|\Gamma\| \vdash_{\mathsf{cc}} |t| : |A'|^\bullet$.

First note that this implies the statement of the theorem. Indeed, by the claim we have $\|\Gamma\| \vdash_{\mathsf{cc}} |t| : |B|^\bullet$ for some $B \in \Lambda_o^\bullet$ with $B \equiv \mathrm{El}_{\underline{n}}\ A$. Therefore $|B|^\bullet \equiv |A|^\bullet = |A|$, so we conclude $\|\Gamma\| \vdash_{\mathsf{cc}} |t| : |A|$ by the conversion rule.

We proceed with the proof of the claim, by induction on t, following the definition of Λ_o. We illustrate the interesting case of λ-abstraction: $t = \lambda x : \mathrm{El}_{\underline{n}}\ A_1.u$. By inversion we have $\Gamma \vdash A_1 : \mathrm{U}_{\underline{n}}$ and $\Gamma, x : \mathrm{El}_{\underline{n}}\ A_1 \vdash u : A_2$ for some A_2 with $A \equiv (x : \mathrm{El}_{\underline{n}}\ A_1) \to A_2$. By i.h. we thus have $\|\Gamma\| \vdash_{\mathsf{cc}} |A_1| : |B_1|^\bullet$ with $B_1 \equiv \mathrm{U}_{\underline{n}}$. Therefore, we have $|B_1|^\bullet \equiv n$, so by conversion we can derive $\|\Gamma\| \vdash_{\mathsf{cc}} |A_1| : n$, and so $\|\Gamma\|, x : |A_1| \vdash_{\mathsf{cc}}$. By i.h. once more, we have $\|\Gamma\|, x : |A_1| \vdash_{\mathsf{cc}} |u| : |B_2|^\bullet$ for some B_2 with $B_2 \equiv A_2$. We can thus derive $\|\Gamma\| \vdash_{\mathsf{cc}} \lambda x : |A_1|.|u| : |(x : \mathrm{El}_{\underline{n}}\ A_1) \to B_2|^\bullet$ and $A \equiv (x : \mathrm{El}_{\underline{n}}\ A_1) \to B_2$. ◀

9 Related work

The first attempt to encode CC in Dedukti dates back to the work of Assaf. He first identified the full-reflection equations (discussed in Section 4) in earlier work studying a variant of the calculus of constructions with explicit cumulativity [4]. There, cumulativity is made explicit by a family of lifts $\uparrow_i : \mathsf{U}_i \to \mathsf{U}_{i+1}$, which are sufficient in his setting because the theory considered lacks product covariance.

These ideas were then employed in encoding a class of cumulative type systems (CTSs) in Dedukti [5], containing in particular the type system CC. In order to handle product covariance, he proposed the use of η-expansion at translation time: for instance, a variable $f : \mathsf{Nat} \to 0$ would be translated *at type* $\mathsf{Nat} \to 1$ as $\lambda x.\uparrow_{\underline{0}}\ (f\ x)$. This however turned out to invalidate conservativity, as observed by Thiré [45, Example 6.6].

Moreover, as mentioned in the introduction, the translation functions used by Assaf for stating and proving soundness turn out to be ill-defined. He mutually defines functions $[-]_\Gamma$ and $[-]_{\Gamma \vdash C}$ and $[\![-]\!]$, and among their defining clauses he states $[t]_{\Gamma \vdash C} := \lambda x : [\![A]\!].[t\ x]_{\Gamma, x:A \vdash B}$ if $C \equiv \Pi x : A.B$ and t has a principal type convertible to $\Pi x : A.B'$ with $B' \subsetneq B$. However, the term A is only determined up to conversion, yet the function is defined over unquontiented terms, and the preservation of conversion is only shown at a later stage. Worse, because A is recovered using typing information, it might not be structurally smaller than t, and no well-founded order is given to justify the recursive call of $[\![-]\!]$ on A.

Regarding confluence, Assaf actually relies in his presentation on an axiomatization of the conversion relation required for the encoding. Because in Dedukti the conversion must be implemented by rewrite rules, each instantiation of his encoding then also needs to provide a rewrite system correctly implementing these equational axioms. In the particular case of CC, Assaf provides rules for implementing them, yet they are not confluent since some critical pairs are not joinable. This problem was later fixed in his joint work with Dowek, Jouannaud and Liu [8], though it required the use of rewriting modulo ACU, which is less efficient and harder to implement than pure syntactic matching. The problems with soundness and conservativity remained unaddressed.

Some years after the work of Assaf, the problem regained attention and new encodings were proposed by Thiré [45], also supporting a class of CTSs, and Férey [26], also supporting universe polymorphism. Starting from Thiré's observation that η-expanding at translation time breaks conservativity, they decided to instead rely on a generalized cast operator mapping a term $t : \mathrm{El}_{l_a}\ a$ to ${}^{l_b}_{l_a}\uparrow^b_a\ e\ t : \mathrm{El}_{l_b}\ b$, where e is a term witnessing the inclusion of a in b. Unfortunately, the use of a multi-step lift then required non-left-linear rules to ensure that two consecutive casts can be composed or that identity casts can be removed. Despite the impressive work of Férey on confluence criteria for non-left-linear systems [25], they were unable to show the confluence of their encodings.

The translation function employed by Thiré unfortunately inherited the issue of Assaf's function, as it also makes recursive calls on terms obtained through typing information without giving a decreasing measure. The proposal of Férey uses however a different technique, and instead defines the translation function over typing derivations. However, because a typing judgment can be derived in multiple ways, he then needs to show that they are nevertheless translated to convertible terms in Dedukti, which corresponds in our case to the coherence property. To show this, he crucially relies on the way terms are represented in his encoding, as untyped codes annotated with a type. Because of this, it is not clear to us how his technique could be adapted to our case, which is why we chose to prove soundness using an inverse translation function, instead of defining a translation function over derivations. Finally, conservativity is stated only as a conjecture for both of Thiré's and Férey's encodings.

10 Conclusion

In this work we have given an encoding of CC in Dedukti satisfying the necessary properties for being used in practice, solving a longstanding open problem. Our proof of confluence combines many confluence criteria and heavily uses the automated tools developed by the community. Yet, at the present moment, none of the available tools are able to fully show our result by themselves. Proving the confluence of our system automatically can thus be an interesting challenge for the next generation of today's confluence checkers. A natural direction could be trying to automate Dowek et al.'s criterion [21], which is the only one we are aware of that can show the confluence of $\beta\mathcal{R}_1$ directly.

Our work has also identified a problem with the definition of the translation function in some previous attempts at encoding CC in Dedukti. To solve this issue, we have then contributed an adaptation of the technique of Winterhalter et al. [48] in which soundness is instead stated and proved using an inverse translation function.

Regarding conservativity, we have proven a restricted form concerning only object terms. Even though we believe that for practical needs our result is sufficient, we conjecture that full conservativity can be obtained by adapting the logical relations technique of Assaf [6]. Alternatively, we could modify our encoding and employ the technique described by Felicissimo [22], which allows for easy conservativity proofs at the cost of increasing the amount of type annotations in the syntax. There is already ongoing work on removing these annotations by incorporating bidirectional typing into Dedukti [23], yet the encoding presented here would not be covered by the presently available framework.

Finally, we believe that our work can be a starting point for incorporating Coq's universe-polymorphism. Among previous work, only Férey considers the combination of CC with universe polymorphism. Combining his ideas with ours is a promising direction to explore.

References

1 AProVE. URL: `https://aprove.informatik.rwth-aachen.de/`.

2 CeTA. URL: `http://cl-informatik.uibk.ac.at/software/ceta/`.

3 CSIho. URL: `http://cl-informatik.uibk.ac.at/software/csi/ho/`.

4 Ali Assaf. A calculus of constructions with explicit subtyping. In *20th International Conference on Types for Proofs and Programs (TYPES 2014)*, volume 39, 2014.

5 Ali Assaf. *A framework for defining computational higher-order logics*. These, École polytechnique, September 2015. URL: `https://pastel.archives-ouvertes.fr/tel-01235303`.

6 Ali Assaf. Conservativity of Embeddings in the lambda Pi Calculus Modulo Rewriting. In Thorsten Altenkirch, editor, *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*, volume 38 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31–44, Dagstuhl, Germany, 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.TLCA.2015.31`.

7 Ali Assaf, Guillaume Burel, Raphaël Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant, and Ronan Saillard. Dedukti: a logical framework based on the λΠ-calculus modulo theory. Unpublished, 2016.

8 Ali Assaf, Gilles Dowek, Jean-Pierre Jouannaud, and Jiaxiang Liu. Untyped Confluence In Dependent Type Theories. working paper or preprint, April 2017. URL: `https://hal.inria.fr/hal-01515505`.

9 H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics. Elsevier Science, 1984.

10 Bruno Barras. Auto-validation d'un systeme de preuves avec familles inductives. *These de doctorat, Université Paris*, 7, 1999.

11 M. Bezem, J.W. Klop, R. de Vrijer, and Terese. *Term Rewriting Systems*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2003. URL: `https://books.google.fr/books?id=7QQ5u-4tRUkC`.

12 Frédéric Blanqui. *Théorie des types et réécriture. (Type theory and rewriting)*. PhD thesis, University of Paris-Sud, Orsay, France, 2001. URL: `https://tel.archives-ouvertes.fr/tel-00105522`.

13 Frédéric Blanqui. Type safety of rewrite rules in dependent types. In *5th International Conference on Formal Structures for Computation and Deduction*, 2020.

14 Frédéric Blanqui. Encoding type universes without using matching modulo AC. In *Proceedings of the 7th International Conference on Formal Structures for Computation and Deduction*, Leibniz International Proceedings in Informatics 228, 2022. `doi:10.4230/LIPIcs.FSCD.2022.24`.

15 Frédéric Blanqui. HOL-Light library in Coq. URL: `https://github.com/Deducteam/coq-hol-light`.

16 Frédéric Blanqui, Gilles Dowek, Emilie Grienenberger, Gabriel Hondet, and François Thiré. A modular construction of type theories. *Logical Methods in Computer Science*, Volume 19, Issue 1, February 2023. `doi:10.46298/lmcs-19(1:12)2023`.

17 Valentin Blot, Gilles Dowek, and Thomas Traversié. An Implementation of Set Theory with Pointed Graphs in Dedukti. In *LFMTP 2022 - International Workshop on Logical Frameworks and Meta-Languages : Theory and Practice*, Haïfa, Israel, August 2022. URL: `https://inria.hal.science/hal-03740004`.

18 Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications*, pages 102–117, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

19 Deducteam. An encoding of impredicativity, cumulativity and product covariance in the logical framework dedukti. URL: `https://github.com/Deducteam/cc-in-dk`.

20 Deducteam. Pull request for ACU matching in DkCheck. URL: `https://github.com/Deducteam/Dedukti/pull/219`.

21 Gilles Dowek, Gaspard Férey, Jean-Pierre Jouannaud, and Jiaxiang Liu. Confluence of left-linear higher-order rewrite theories by checking their nested critical pairs. *Mathematical Structures in Computer Science*, 32(7):898–933, 2022. `doi:10.1017/S0960129522000044`.

22 Thiago Felicissimo. Adequate and Computational Encodings in the Logical Framework Dedukti. In Amy P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*, volume 228 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.FSCD.2022.25`.

23 Thiago Felicissimo. Generic bidirectional typing for dependent type theories. In Stephanie Weirich, editor, *Programming Languages and Systems*, pages 143–170, Cham, 2024. Springer Nature Switzerland.

24 Thiago Felicissimo, Frédéric Blanqui, and Ashish Kumar Barnawal. Translating Proofs from an Impredicative Type System to a Predicative One. In Bartek Klin and Elaine Pimentel, editors, *31st EACSL Annual Conference on Computer Science Logic (CSL 2023)*, volume 252 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:19, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CSL.2023.19`.

25 Gaspard Férey and Jean-Pierre Jouannaud. Confluence in non-left-linear untyped higher-order rewrite theories. In *Proceedings of the 23rd International Symposium on Principles and Practice of Declarative Programming*, PPDP '21, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3479394.3479403.

26 Gaspard Férey. *Higher-Order Confluence and Universe Embedding in the Logical Framework*. These, Université Paris-Saclay, June 2021. URL: https://tel.archives-ouvertes.fr/tel-03418761.

27 Jan Herman Geuvers. *Logics and type systems*. [Sl: sn], 1993.

28 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Automated termination proofs with AProVE. In *Rewriting Techniques and Applications: 15th International Conference, RTA 2004, Aachen, Germany, June 3-5, 2004. Proceedings 15*, pages 210–220. Springer, 2004.

29 Yoan Géran. Mathématiques inversées de Coq, 2021. URL: https://inria.hal.science/hal-04319183.

30 Makoto Hamana. How to prove your calculus is decidable: Practical applications of second-order algebraic theories and computation. *Proc. ACM Program. Lang.*, 1(ICFP), August 2017. doi:10.1145/3110266.

31 Hugo Herbelin. Type inference with algebraic universes in the calculus of inductive constructions. *Unpublished. Available at: pauillac. inria. fr/ herbelin/publis/univalgcci. pdf*, 2005.

32 Gabriel Hondet and Frédéric Blanqui. Encoding of Predicate Subtyping with Proof Irrelevance in the Lambdapi-Calculus Modulo Theory. In Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch, editors, *26th International Conference on Types for Proofs and Programs (TYPES 2020)*, volume 188 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.TYPES.2020.6.

33 Jan Willem Klop. *Combinatory reduction systems*. PhD thesis, Rijksuniversiteit Utrecht, 1963.

34 Marc Lasson. *Réalisabilité et paramétricité dans les systèmes de types purs*. Theses, Ecole normale supérieure de lyon - ENS LYON, November 2012. URL: https://theses.hal.science/tel-00770669.

35 Zhaohui Luo. *An extended calculus of constructions*. PhD thesis, University of Edinburgh, 1990.

36 Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical computer science*, 192(1):3–29, 1998.

37 Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of logic and computation*, 1(4):497–536, 1991.

38 Julian Nagele, Bertram Felgenhauer, and Aart Middeldorp. CSI: New evidence — a progress report. In Leonardo de Moura, editor, *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *Lecture Notes in Artificial Intelligence*, pages 385–397, 2017. doi:10.1007/978-3-319-63046-5_24.

39 Nicolas Oury. Extensionality in the calculus of constructions. In *Theorem Proving in Higher Order Logics: 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005. Proceedings 18*, pages 278–293. Springer, 2005.

40 Ronan Saillard. Rewriting modulo β in the $\lambda\pi$-calculus modulo. *Electronic Proceedings in Theoretical Computer Science*, 185:87–101, July 2015. doi:10.4204/eptcs.185.6.

41 Ronan Saillard. *Type checking in the Lambda-Pi-calculus modulo: theory and practice*. PhD thesis, Mines ParisTech, France, 2015. URL: https://pastel.archives-ouvertes.fr/tel-01299180.

42 The Coq Development Team. Typing rules for Coq. URL: https://coq.inria.fr/doc/V8.16.1/refman/language/cic.html#id6.

43 René Thiemann and Christian Sternagel. Certification of termination proofs using ceta. In *International Conference on Theorem Proving in Higher Order Logics*, pages 452–468. Springer, 2009.

44 François Thiré. Sharing a library between proof assistants: Reaching out to the HOL family. In Frédéric Blanqui and Giselle Reis, editors, *Proceedings of the 13th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP@FSCD 2018, Oxford, UK, 7th July 2018*, volume 274 of *EPTCS*, pages 57–71, 2018. doi:10.4204/EPTCS.274.5.

45 François Thiré. *Interoperability between proof systems using the logical framework Dedukti*. PhD thesis, ENS Paris-Saclay, 2020.

46 Vincent van Oostrom. Developing developments. *Theoretical Computer Science*, 175(1):159–181, 1997.

47 Vincent van Oostrom and Femke van Raamsdonk. Weak orthogonality implies confluence: The higher-order case. In Gerhard Goos, Juris Hartmanis, Anil Nerode, and Yu. V. Matiyasevich, editors, *Logical Foundations of Computer Science*, volume 813, pages 379–392. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994. Series Title: Lecture Notes in Computer Science. doi:10.1007/3-540-58140-5_35.

48 Théo Winterhalter, Matthieu Sozeau, and Nicolas Tabareau. Eliminating Reflection from Type Theory. In *CPP 2019 – 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 91–103, Lisbonne, Portugal, January 2019. ACM. doi:10.1145/3293880.3294095.