HAL open science

Coq without Type Casts: A Complete Proof of Coq Modulo Theory

Jean-Pierre Jouannaud, Pierre-Yves Strub

To cite this version:

Jean-Pierre Jouannaud, Pierre-Yves Strub. Coq without Type Casts: A Complete Proof of Coq Modulo Theory. LPAR-21: 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, May 2017, Maun, Botswana. hal-01664457

HAL Id: hal-01664457

https://inria.hal.science/hal-01664457v1

Submitted on 14 Dec 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Coq without Type Casts: A Complete Proof of Coq Modulo Theory

Jean-Pierre Jouannaud[1,2] and Pierre-Yves Strub[2]

[1] INRIA, Project Deducteam, Université Paris-Saclay, France
[2] LIX, École Polytechnique, France

Abstract

Incorporating extensional equality into a dependent intensional type system such as the Calculus of Constructions provides with stronger type-checking capabilities and makes the proof development closer to intuition. Since strong forms of extensionality lead to undecidable type-checking, a good trade-off is to extend intensional equality with a decidable first-order theory T, as done in CoqMT, which uses matching modulo T for the weak and strong elimination rules, we call these rules T-elimination. So far, type-checking in CoqMT is known to be decidable in presence of a cumulative hierarchy of universes and weak T-elimination. Further, it has been shown by Wang with a formal proof in Coq that consistency is preserved in presence of weak and strong elimination rules, which actually implies consistency in presence of weak and strong T-elimination rules since T is already present in the conversion rule of the calculus.

We justify here CoqMT's type-checking algorithm by showing strong normalization as well as the Church-Rosser property of β-reductions augmented with CoqMT's weak and strong T-elimination rules. This therefore concludes successfully the meta-theoretical study of CoqMT.

Acknowledgments: to the referees for their careful reading.

1 Introduction

Type checking for an extensional type theory being undecidable, as is well-known since the early days of Martin-Löf's type theory [16], most proof assistants implement an intensional type theory. Coq [21] implements CIC$^\omega$, a type theory that extends the Calculus of Constructions of Coquand and Huet (CC [10]) with inductive types in the style of the Calculus of Inductive Constructions of Coquand and Paulin-Mohring (CIC [18]), and a predicative, cumulative hierarchy of universes in the style of the Extended Calculus of Constructions of Luo (ECC [15]), as reformulated by Werner in [23].

The purely intensional version of type-theory may become awkward when it comes to programming with dependent types. In the well-known example of "vectors", one has the type $\mathsf{Vect}(n)$ of lists of length n, a concatenation function @ such that $v_1@v_2$ has length $n_1 + n_2$ whenever v_1 and v_2 have length n_1 and n_2 respectively. But showing that $v@\mathsf{nil} = v$ is not possible because it is not even a well-typed statement: lengths $n + 0$ and n are not identified in Coq whenever $+$ is defined recursively on the first argument (n here), which must therefore be in the form of 0 or successor of something to generate the expected computation.

There are many ways to fix the problem: by adding rewrite rules to increase the computing ability [7]; by building in full extensionality to the price of loosing decidability [19, 17]; by building in a restricted form of extensionality taking advantage of equations present in the typing context [8]; and by building in an even weaker form of extensionality restricted to an object level first-order theory [20]. The difficulty is indeed to find the right trade-off between the expressivity of the type system, the decidability of type-checking, and the efficiency of the

implementation. Among all these proposals, the last one only has been implemented as a proof assistant named CoqMT for Coq Modulo Theory [20]. The formal theory underlying CoqMT was fully described under the name of CoqMTU in [6]. An original feature of CoqMTU was its weak elimination rule using pattern matching modulo the first-order theory T, called T-elimination. In our example of Presburger arithmetic, this allows to pop up zero or successor from a expression headed by $+$, regardless of the inductive definition of $+$. CoqMTU lacked strong elimination which we were unable to justify at that time.

We then started a program aiming at a full formal proof of $\mathrm{CIC}^{\omega}(\mathrm{T})$ (ω standing for the cumulative hierarchy of predicative universes), our now favorite name for the theory obtained by adding strong T-elimination to CoqMTU. Proving Coq in Coq is an effort started long ago by Barras, who completed it recently with a formal, semantic proof of strong normalization and consistency of CIC^{ω} [4]. A major step in this success was Werner's encoding of type theory in Zermelo Fraenkel intuitionistic set theory IZF augmented by arbitrarily many inaccessible cardinals [23], further improved by Barras [5]. This opened the way to a full proof of the meta-theory of CIC^{ω}, the theory underlying Coq, for which $T = \emptyset$, via the elaboration of a complex, structured, semantic model, building upon previous work of Altenkirch [1]. A major advantage of Barras' Coq development is its modularity: starting with an appropriate realizability model of CC based on IZF augmented with inaccessible cardinals, adding new semantic objects for each new syntactic construct, and proving conservativity of the considered extension shows then both that the obtained calculus is consistent, and also that the reductions are strongly-normalizing in the extension. This approach has been taken for CC augmented with a predicative hierarchy of universes first (CC^{ω}), then with inductive types (CIC^{ω}). It can be carried out smoothly, provided successive extensions do not interact.

Applied to $\mathrm{CIC}^{\omega}(\mathrm{T})$, this modular approach has allowed Wang to get consistency and strong normalization of a restriction of $\mathrm{CIC}^{\omega}(\mathrm{T})$ for which elimination is classically fired via plain pattern matching rather than matching modulo the theory T [22]. The reason is that T-elimination would make two different extensions depend on each other: inductive types and T. But since the theory T is present via the conversion rule of $\mathrm{CIC}^{\omega}(\mathrm{T})$, this modular approach shows indeed consistency of the whole calculus $\mathrm{CIC}^{\omega}(\mathrm{T})$ as well as strong normalization of β-reduction augmented with the classical plain elimination rules. This does not, however, show strong normalization when including T-elimination, and therefore, the decidability of type-checking in $\mathrm{CIC}^{\omega}(\mathrm{T})$ does not follow from [22].

The crux of the problem is indeed to show that T-elimination and β-reduction together are strongly normalizing. Our approach relies upon the strong normalization property of classical plain elimination together with β-reduction used as an induction principle in order to derive, by syntactic arguments, the strong normalization property of β-reduction augmented with T-elimination. This syntactic strong normalization proof, which then allows to show decidability of type-checking by standard arguments, is our main contribution. Done *outside* Barras' model construction, it shows that his expandable semantic infrastructure forbidding interactions between different semantic objects, can be extended again by solving interactions between different semantic objects at the syntactic level.

Syntax and semantics of $\mathrm{CIC}^{\omega}(\mathrm{T})$ are given in Section 2. Our syntactic strong normalization proof is developed in Section 3 as well as the proof of the Church-Rosser property. An extended example follows in Section 4.

2 Syntax of $\mathrm{CIC}^{\omega}(\mathrm{T})$

Apart from cosmetic differences, this section follows the presentation given in [6].

The main novel feature of $\mathrm{CIC}^\omega(\mathrm{T})$ and its predecessors with respect to CIC^ω is the embedding of a type of first-order terms. Regarding the typing rules, the main difference is the extension of the conversion relation by a decidable congruence $\sim_T$ on these first-order terms homomorphically extended to arbitrary terms. Regarding now the computation rules, the main difference is the use of pattern matching in the theory for firing the so-called T-elimination rules. There are different version of $\mathrm{CIC}^\omega(\mathrm{T})$ incorporating different mechanisms.[1]

The present version of $\mathrm{CIC}^\omega(\mathrm{T})$ does have strong elimination, which is witnessed by the fact that the eliminator of natural numbers $\textsc{Elim}_{\mathsf{Nat}}(\mathsf{P},\mathsf{u},\mathsf{v},\mathsf{t})$, in the case of Presburger arithmetic, can be used with a term P belonging to any sort s. Further, T-eliminations incorporate the theory $\sim_{\mathcal{T}}$ into reductions via pattern matching in the theory, which must therefore be assumed to be decidable. $\mathrm{CIC}^\omega(\mathrm{T})$ appears then as a generalization of CIC^ω, the latter being an instance of the former in case the theory $\sim_T$ is syntactic equality.

The version of $\mathrm{CIC}^\omega(\mathrm{T})$ introduced in [6] under the name CoqMTU, has weak T-elimination only, obtained by restricting s to the sort of propositions. As a consequence, CoqMTU provided the induction principle of the natural numbers, but no possibility to define functions by structural induction on them.

The version of $\mathrm{CIC}^\omega(\mathrm{T})$ studied in [22] is intermediate: there are weak and strong elimination rules, but the theory $\sim_T$ is not incorporated into reductions. Further, it uses a direct judgmental presentation, instead of lifting the first-order equalities into equality judgments on terms via the embedding of the type of first-order terms. The equivalence of both presentations is formally proved in [4] in a restricted case, but does hold, as expected, in the general case.

2.1 First-Order Theory T

We consider a first-order mono-sorted algebra defined by a *sort* o, a non-empty set of *constructor symbols* $\mathcal{C}$, a set of *defined symbols* $\mathcal{D}$, both equipped with arities, and a set of *variables* $\mathcal{X}$. Multi-sorted algebras would do as well. We denote by $\mathcal{T}(\mathcal{C}\uplus\mathcal{D},\mathcal{X})$ (or simply $\mathcal{T}$) the set of (first-order) terms, $\mathcal{T}(\mathcal{C},\mathcal{X})$ the set of constructor terms, $\mathcal{T}(\mathcal{D},\mathcal{X})$ the set of defined terms, and drop the letter $\mathcal{X}$ for the respective sets of *ground* terms. Constructors and defined symbols are equipped with a fixed *arity*, denoted by $\mathrm{arity}(f)$ for the symbol f.

The semantics of the defined symbols is specified in an abstract form by a *decidable congruence* $\sim_T$ over $\mathcal{T}(\mathcal{C}\uplus\mathcal{D},\mathcal{X})$, the so-called theory T. Terms s, t such that $s\sim_T t$ are called T*-convertible*. Congruences are extended to n-tuples of terms as expected. We assume that T satisfies the following axioms:

Freeness. For all constructors C, C' and terms $\overline{u},\overline{v}$, $C(\overline{u})\sim_T C'(\overline{v})$ implies $C=C'$ and $\overline{u}\sim_T\overline{v}$.

When the congruence $\sim_T$ is generated by a set of equations, freeness is usually ensured by constraining the generating equations (assuming for example confluence and that the equations are oriented into rules whose left-hand sides are not constructor-headed). It is not enough to weaken the present assumption by assuming that equivalent constructor terms are syntactically equal, as pointed out to us by Jesper Cockx.[2]

Non-triviality. $\mathcal{T}(\mathcal{C})$ contains at least two different constructor terms.

Under the freeness assumption, this second assumption is equivalent to the existence of at least two constructors. It will play an important role later.

[1] We reserve the acronym CoqMT for the implementation incorporating them all.

[2] Who provided us with the following counter-example to Lemma II.C.4 in [6]: let $\mathcal{C}\stackrel{\text{def}}{=}\{0,S\},\mathcal{D}\stackrel{\text{def}}{=}\{f,g\}$ and $S(f(x))\sim_T S(g(x)), f(0)\sim_T 0, g(0)\sim_T 0, f(S(0))\sim_T 0, g(S(0))\sim_T 0$, but $f(x)\not\sim_T 0$ and $g(x)\not\sim_T 0$. Then $S(f(x))\sim_T S(g(x))$, but $f(x)\not\sim_T g(x)$.

Completeness. For all terms t in $\mathcal{T}(\mathcal{C} \uplus \mathcal{D})$, there exists a term u in $\mathcal{T}(\mathcal{C})$ (unique by *Freeness*) such that $t \sim_T u$.

Completeness ensures that the quotient $T(\mathcal{C} \uplus \mathcal{D})/\sim_T$ is isomorphic to $T(\mathcal{C})$. Although this is undecidable, good tools exist such as SPIKE [9].

Example 2.1. *Let a, c be constructors with arities $0, 1$, and f a defined symbol of arity 0. Let T the theory defined by the equation $f = c(f)$. T is decidable, since the rewrite system $c(f) \to f$ defines unique normal forms, satisfies freeness and non-triviality, but is not complete: no* finite *constructor term is equivalent to f.*

Our paradigmatic first-order theory satisfying our axioms is Presburger Arithmetic, whose alphabet is $\mathcal{C} = \{\mathbf{0}, \mathbf{S}\}$ and $\mathcal{D} = \{+\}$. We indeed use here the unquantified fragment of Presburger arithmetic, but could take the unquantified fragment of Peano arithmetic as well, since that fragment is decidable.

2.2 Pseudo-Terms

Since our abstract calculus contains the calculus of constructions, universes, inductive types, and a first-order theory $\sim_T$, its term language contains the usual term constructions of CC, universes, eliminators, and terms from the first-order language. Incorporating the latter into the type-theoretic language is easily done by declaring the first-order function symbol as higher-order constants in the calculus.

Example 2.2. *In the example of Presburger arithmetic, this gives $\mathbf{0} : \mathbf{nat}, \mathbf{S} : \mathbf{nat} \to \mathbf{nat}$, and $+ : \mathbf{nat} \to \mathbf{nat} \to \mathbf{nat}$. Then,* fully applied *terms (like $(+\ \mathbf{0}\ \mathbf{0})$) correspond to first-order terms (here, $+\ (\mathbf{0}, \mathbf{0})$), while non-fully applied ones like $(+\ \mathbf{0})$ do not.*

Universes: our universes are classically **Prop** and $\mathbf{Type}_j$, where j is a (strictly) positive integer. We shall identify **Prop** with $\mathbf{Type}_0$ whenever convenient. This goes against the Coq tradition of identifying **Prop** with $\mathbf{Type}_{-1}$ and **Set** with $\mathbf{Type}_0$, but is a natural fit with the predicativity of **Set**. Then, $s := \{\mathbf{Type}_j\}_{j \geq 0}$.

Variables are elements of a denumerable set $\mathcal{V}$ containing $\mathcal{X}$ as a denumerable subset.

First-order constants: we denote by o the type of our first-order algebraic expressions, and (abusing notations) by $\mathcal{C}$ and $\mathcal{D}$ the sets of higher-order constants corresponding to the constructors and defined symbols respectively.

Eliminator: we denote by $\textsc{Elim}_o$ the eliminator for the type o.

Pseudo terms:

$$t, u, P, Q ::= s \mid o \mid \mathcal{C} \mid \mathcal{D} \mid \mathcal{V} \mid t\ u \mid \lambda[x \in \mathcal{X} : P].\, t \mid \forall[x \in \mathcal{X} : P].\, Q \mid \textsc{Elim}_o(P, \overline{u}, t).$$

As usual, application associates to the left. We will systematically use the left form of a sequence of applications. Positions in terms are defined as usual, using Λ for the root position. We denote by $\mathcal{FP}\mathrm{os}(t)$ the set of non-variable positions in the term t, by $t(p)$ the symbol at position p in t –$t(\Lambda)$ being the *head* of t–, by $\mathcal{V}\mathrm{ar}(t)$ the set of free variables of a term t, and by $|t|$ its size, counting then the non-variable nodes of its tree representation.

Example 2.3. *For Presburger arithmetic, $\textsc{Elim}_{Nat}(P, u, v, t)$ stands for Gödel's recursor at higher type $rec(P, t, u, v)$.*

Pseudo substitutions: A (pseudo) *substitution* of *domain* $\mathcal{D}\mathrm{om}(\theta) = \{x_1, \ldots, x_n\}$ is a sequence $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where the x_i's are distinct variables and the t_i's are terms. A substitution θ acts on a term u by replacing all free occurrences of the variables x_i's in u by the corresponding t_i's, possibly renaming bound variables.

2.3 Embedding the Algebraic World

Definition 2.4. *A pseudo-term is* algebraic-headed *if it is of the form* $(f\, t_1 \ldots t_n)$ *where* $f \in \mathcal{C} \uplus \mathcal{D}$ *and* $\mathrm{arity}(f) = n$*. We also say that* f *is fully applied. It is* algebraic *if it is a variable, or an algebraic-headed term* $(f\, t_1 \ldots t_n)$ *whose pseudo-terms* $t_1, \ldots, t_n$ *are algebraic.*

Algebraic pseudo-terms are identified with first-order terms in $\mathcal{T}(\mathcal{C} \uplus \mathcal{D}, \mathcal{V})$.

Definition 2.5. *A pseudo-term is* curried *if none of its (non-strict) subterms is algebraic headed.*

For an example, if f has arity 2, then $(f\ \lambda x.x)$ is curried.

Definition 2.6. *Algebraic and curried terms are said to be* homogeneous*, others are* heterogeneous*. Given a term* t*, an* immediate alien *is any maximal strict subterm* u *of* t *such that*

- *if* t *is algebraic headed, then* u *is not algebraic headed*
- *if* t *is not algebraic headed, then* u *is algebraic headed.*

We denote by $\mathcal{A}(t)$ *the multi-set of aliens of* t*.*

Note that the head and immediate subterms (or super-terms) of a term must be defined properly: the head and immediate subterms of $(\ldots(ft_1)\ldots t_n)$ are f and all the t_i's in case $n = \mathrm{arity}(f)$, but the application operator, $(\ldots(ft_1)\ldots t_{n-1})$ and t_n otherwise.

We now decompose an arbitrary term s into an *homogeneous cap* $\widehat{s}$ and an *heterogeneous alien* substitution σ such that $s = \widehat{s}\sigma$. To this end, variables in the cap are chosen according to a one to one mapping ξ from classes of pseudo-terms modulo renaming of bound variables and the set $\mathcal{V} \setminus \mathcal{X}$ of so-called *fresh* variables.

Definition 2.7 ([6])**.** *Given a pseudo-term* t*, we define its* cap $\widehat{t}$ *and* alien substitution θ_t *as:*

(i) $\widehat{t} \stackrel{\text{def}}{=} t[\xi(t|_p)]_{p \in P}$*, where* P *is the set of positions of the immediate aliens of* t*;*

(ii) $\theta_t \stackrel{\text{def}}{=} \{\xi(t|_p) \mapsto t|_p \mid t|_p$ *is an immediate alien of* $t\}$*.*

Note that this definition trivially ensures the property $t = \widehat{t}\theta_t$.

Example 2.8. *Let* $s = (+\ (\lambda x : \mathbf{nat}.(\mathbf{S}\, x)\, \mathbf{0})(+\ (+\ \mathbf{0})(+\ \mathbf{0})))$*. Then,* $\widehat{s} = +\ (y, +\ (z, z))$*,* $y, z \in \mathcal{V} \setminus \mathcal{X}$*, and* $\theta_s = \{y \mapsto (\lambda x : \mathbf{nat}.(\mathbf{S}\, x)\, \mathbf{0}), z \mapsto (+\ \mathbf{0})\}$*.*

Definition 2.9. *Figure 1 extends the congruence* $\sim_T$ *to a decidable conversion relation* $\leftrightarrow_T^*$*.*

Definition 2.10. *The* constructor size $n_{\mathcal{C}}(t)$ *of a term* t *is the maximum of the set*

$$\{|v| \mid\ v \in \mathcal{T}(\mathcal{C}, \mathcal{X}) \text{ and } t \leftrightarrow_T^* v\theta \text{ for some substitution } \theta\}.$$

The relation $\leftrightarrow_T^*$ is transitive [6], explaining our notation. Simpler formulation can be given, the present one eases some proofs by induction. The coming properties are needed for our normalization proof:

$$\frac{u =_\alpha v}{u \leftrightarrow_T^* v} \qquad \frac{\begin{array}{c}\hat{u} \text{ or } \hat{v} \text{ is not a variable}\\ \hat{u} \sim_T \hat{v} \qquad \theta_u \leftrightarrow_T^* \theta_v\end{array}}{\hat{u}\theta_u \leftrightarrow_T^* \hat{v}\theta_v} \qquad \frac{T \leftrightarrow_T^* W \qquad t \leftrightarrow_T^* w}{\lambda[x:T].\,t \leftrightarrow_T^* \lambda[x:W].\,w}$$

$$\frac{T_1 \leftrightarrow_T^* W_1 \qquad T_2 \leftrightarrow_T^* W_2}{\forall[x:T_1].\,T_2 \leftrightarrow_T^* \forall[x:W_1].\,W_2} \qquad \frac{M \leftrightarrow_T^* M' \qquad N \leftrightarrow_T^* N'}{M\ N \leftrightarrow_T^* M'\ N'}$$

$$\frac{Q_1 \leftrightarrow_T^* Q_2 \qquad \overline{f_1} \leftrightarrow_T^* \overline{f_2} \qquad t_1 \leftrightarrow_T^* t_2}{\textsc{Elim}_o(Q_1, \overline{f_1}, t_1) \leftrightarrow_T^* \textsc{Elim}_o(Q_2, \overline{f_2}, t_2)} \qquad \frac{\forall x \in \mathcal{D}\mathrm{om}(\theta_u) \cap \mathcal{D}\mathrm{om}(\theta_v).\,\theta_u(x) \leftrightarrow_T^* \theta_v(x)}{\theta_u \leftrightarrow_T^* \theta_v}$$

Figure 1: Inference rules for $\leftrightarrow_T^*$

Lemma 2.11 ([6])**.** *The constructor size of any term is finite.*

Lemma 2.12 ([6])**.** *Assume $s \leftrightarrow_T^* t$, where $n_{\mathcal{C}}(s) = n_{\mathcal{C}}(t) > 0$.*

(i) There exists $u = C(\overline{u})$ such that C is a constructor, $s \leftrightarrow_T^* u$, $t \leftrightarrow_T^* u$, and $\mathcal{V}\mathrm{ar}(u) \subseteq \mathcal{V}\mathrm{ar}(s) \cap \mathcal{V}\mathrm{ar}(t)$.

(ii) Let $s \leftrightarrow_T^* C(\overline{u})$ and $t \leftrightarrow_T^* C'(\overline{v})$. Then $C = C'$ and $\overline{u} \leftrightarrow_T^* \overline{v}$.

Definition 2.13. *A term $C(\overline{u})$ simplifies a term t iff (i) $t = C(\overline{u})$, or (ii) $t(\Lambda) \notin \mathcal{C}$, $t \leftrightarrow_T^* C(\overline{u})$ and $\mathcal{V}\mathrm{ar}(\overline{u}) \subseteq \mathcal{V}\mathrm{ar}(t)$.*

This definition strengthens the one given in [6] by assuming that $C(\overline{u})$ is t itself in case it is constructor-headed. The usual elimination rule for inductive types becomes then a particular case of the T-elimination rule of CIC$^\omega$(T). This modification also eases the computation of $C(\overline{u})$ as well as some proofs to come.

Lemma 2.14 ([6])**.** *Assume $C(\overline{u})$ simplifies t. Then, $n_{\mathcal{C}}(t) > n_{\mathcal{C}}(\overline{u})$.*

Lemma 2.15. *Let t be a term of a strictly positive constructor size. Then, there exists $\overline{v}$ such that $C(\overline{v})$ simplifies t.*

Proof. If $t = C(\overline{u})$, we are done. Otherwise, $t \leftrightarrow_T^* C(\overline{u})$. The property then follows from Lemma 2.12 (i). □

2.4 Reduction

There are three kinds of reduction in our calculus: β-reduction ($\rightarrow_\beta$), ι-reduction ($\rightarrow_\iota$), and ι_T-reduction ($\rightarrow_{\iota_T}$), whose union is denoted by $\longrightarrow_{\beta\iota_T}$. These reductions are generated by the following rewrite rules:

[β reduction] $(\lambda[x:T].\,u)t \rightarrow_\beta u\{x \mapsto t\}$.

[ι reduction] $\rightarrow_\iota$ is the same as in CIC. It is reserved for *big* inductive types - inductive types whose constructors take functional arguments - which cannot be declared as a first-order algebra equipped with a (possibly trivial) decidable theory T.

[ι_T reduction] $\rightarrow_{\iota_T}$ generalizes plain ι-reduction in the sense that the latter is a particular case of the former, for (small) inductive types whose constructors are first-order, when T is trivial.

We oftentimes superscript rewrite relations by the position of the rewritten redex.

Example 2.16. *For our example of Presburger arithmetic, we have:*

$$\textsc{Elim}_{Nat}(Q, f_{\mathbf{0}}, f_{\mathbf{S}}, t) \to_{\iota_T} \begin{cases} f_{\mathbf{0}} & \textit{if } \mathbf{0} \textit{ simplifies } t \\ f_{\mathbf{S}}\ u\ \textsc{Elim}_{Nat}(Q, f_{\mathbf{0}}, f_{\mathbf{S}}, u) & \textit{if } \mathbf{S}\ u \textit{ simplifies } t \end{cases}$$

With the traditional elimination rule, t is identical to $\mathbf{S}\ u$, and therefore, u has a smaller size and is typable when t is typable. Our requirement that $\mathbf{S}\ u$ simplifies t is essential: it ensures that u has a strictly smaller constructor size, and that it is again typable when t is. Furthermore, Lemma 2.15 ensures that such a term exists as soon as t is T-equivalent to a term headed by a fully applied constructor.

In the sequel, we assume for simplicity of notations that all inductive types are given as algebraic types equipped with a decidable equational theory, the trivial one for traditional (small) inductive types. Accommodating big inductive types in the traditional way is no challenge: the whole meta-theory including strong-normalization is the same, provided *all* inductive types are at the propositional level. This is *not* the case for inductive types defined at the predicative level, since proof-irrelevant interpretations cannot be used anymore for proving strong normalization.

Notations we use the notations: $\to$ for $\longrightarrow_{\beta\iota_T}$, $\leftarrow$ for the inverse of $\to$, $\to^*$ for its reflexive, transitive closure, $\leftrightarrow^*$ for its symmetric, reflexive, transitive closure, and $\simeq$ for the *conversion* relation defined as $(\leftrightarrow_T^* \cup \leftrightarrow^*)^*$. Reductions and the like are extended to substitutions as expected.

2.5 Typing

An environment Γ is a sequence of pairs made of a (fresh) variable and a pseudo-term. We denote by $\mathcal{D}\mathrm{om}(\Gamma) = \{x_i \mid x_i : T_i \in \Gamma\}$ the *domain* of the environment Γ. We often consider environments as substitutions, writing $x\Gamma = T$ if $x : T \in \Gamma$. An environment Δ *contains* an environment Γ, written $\Gamma \subseteq \Delta$, if all pairs in Γ appear in Δ in the same order. An environment Γ' *simplifies* an environment Γ if T' simplifies T for some pairs $x : T \in \Gamma$ and $x : T' \in \Gamma'$. An environment Γ *reduces* to an environment Γ if T reduces to T' for some pairs $x : T \in \Gamma$ and $x : T' \in \Gamma'$. Two environments Γ, Γ' are *compatible* modulo a relation $\mathcal{R}$ on pseudo-terms if for any $x \in \mathcal{D}\mathrm{om}(\Gamma) \cap \mathcal{D}\mathrm{om}(\Gamma')$, $(x\Gamma)\ \mathcal{R}\ (x\Gamma')$.

Our typing rules given at Figure 2 come in three parts: rules for CC, rules for the universes, and rules for the first-order symbols. These rules are the same for CoqMTU and CIC$^\omega$(T). We recall below the key result of [22]:

Theorem 2.17 ([22])**.** *$\beta\iota$-reductions are terminating, type preserving, and confluent (modulo α) on typable terms.*

Note that typable $\beta\iota T$-conversions are proved to be joinable with $\beta\iota$ modulo $T \cup \alpha$ [22]]. We are not going to need this strong form of confluence here.

Given an arbitrary rewrite relation $\longrightarrow_R$, we denote by $\longrightarrow_{R\rhd}$ the relation $\longrightarrow_R \cup \rhd$. An important property repeatedly used in the sequel is that the union of a terminating, monotonic rewrite relation and strict subterm is terminating [11]. Monotonicity is required, which is the case of any well-behaved rewrite relation, as is the case of $\longrightarrow_{\beta\iota}$, which is terminating on all typable terms by Theorem 2.17. Hence, as a particular case, $\longrightarrow_{\beta\iota\rhd}$ is terminating on all typable terms.

$$[\textsc{Var}]\ \frac{\Gamma \vdash T : \mathbf{Type}_j}{\Gamma, x : T \vdash x : T}\ x \notin \mathcal{D}\mathrm{om}(\Gamma) \qquad [\textsc{Weak}]\ \frac{\Gamma \vdash t : T, \quad \Gamma \vdash V : \mathbf{Type}_j}{\Gamma, x : V \vdash t : T}\ x \notin \mathcal{D}\mathrm{om}(\Gamma)$$

$$[\textsc{Lam}]\ \frac{\Gamma, (x \in \mathcal{X} : U) \vdash t : V \qquad \Gamma \vdash \forall[x : U].\, V : \mathbf{Type}_j}{\Gamma \vdash \lambda[x : U].\, t : \forall[x : U].\, V}$$

$$[\textsc{App}]\ \frac{\Gamma \vdash u : \forall[x : U].\, V \qquad \Gamma \vdash v : U}{\Gamma \vdash u\ v : V[x \mapsto v]} \qquad [\textsc{Hierarchy}_{j\geq 0}]\ \frac{}{\vdash \mathbf{Type}_j : \mathbf{Type}_{j+1}}$$

$$[\textsc{Cum}_{j\geq 0}]\ \frac{\Gamma \vdash T : \mathbf{Type}_j}{\Gamma \vdash T : \mathbf{Type}_{j+1}} \qquad [\textsc{Imp}]\ \frac{\Gamma \vdash U : \mathbf{Type}_j \qquad \Gamma, x \in \mathcal{X} : U \vdash V : \mathbf{Type}_0}{\Gamma \vdash \forall[x : U].\, V : \mathbf{Type}_0}$$

$$[\textsc{Pred}]\ \frac{\Gamma \vdash U : \mathbf{Type}_i \qquad \Gamma, x \in \mathcal{X} : U \vdash V : \mathbf{Type}_{j\neq 0}}{\Gamma \vdash \forall[x : U].\, V : \mathbf{Type}_{max(i,j)}} \qquad [\mathbf{nat}]\ \frac{}{\vdash \mathbf{nat} : \mathbf{Type}_0}$$

$$[\mathbf{0}]\ \frac{}{\vdash \mathbf{0} : \mathbf{nat}} \qquad [\mathbf{S}]\ \frac{}{\vdash \mathbf{S} : \mathbf{nat} \to \mathbf{nat}} \qquad [+]\ \frac{}{\vdash + : \mathbf{nat} \to \mathbf{nat} \to \mathbf{nat}}$$

$$[\textsc{Conv}]\ \frac{\Gamma \vdash t : U \qquad \Gamma \vdash U' : \mathbf{Type}_j}{\Gamma \vdash t : U'}\ U \simeq U'$$

$$[\textsc{Elim}]\ \frac{\begin{array}{c}\Gamma \vdash t : \mathbf{nat} \qquad \Gamma \vdash P : \forall[x : \mathbf{nat}].\, \mathbf{Type}_0 \\ \Gamma \vdash f_{\mathbf{0}} : P\ \mathbf{0} \qquad \Gamma \vdash f_{\mathbf{S}} : \forall[x : \mathbf{nat}].\, (P\ x \to P\ (\mathbf{S}\ x))\end{array}}{\Gamma \vdash \textsc{Elim}_{Nat}(P, f_{\mathbf{0}},\ f_{\mathbf{S}}, t) : P\ t}$$

Figure 2: CoqMTU typing rules

3 Decidability of Type-Checking in CIC$^\omega$(T)

Decidability of type checking follows from two other properties: termination and confluence of reductions.

We say that a term s is SN ($\longrightarrow$) if there is no infinite $\longrightarrow$-derivation originating from s, and that the relation $\longrightarrow$ is SN, or terminating, if every term is SN ($\longrightarrow$).

To reach our goal, we are going to study $\longrightarrow_{\beta\iota_T}$-reductions issuing from well-typed terms, and show that they are both SN and Church-Rosser, therefore allowing to decide convertibility of well-typed terms. But instead of studying T-elimination directly, we shall actually *encode* it by introducing a new rewrite relation, *T-simplification*, whose union with elimination will contain T-elimination. To this end, we associate to the congruence $\sim_T$ on algebraic terms the rewrite system:

$$\mathcal{T} = \{t \to C(\overline{u})\ :\ C(\overline{u}) \text{ simplifies } t\}$$

By Lemma 2.14, $n_{\mathcal{C}}(t) > n_{\mathcal{C}}(\overline{u}) \geq 0$, hence t cannot be a variable. Further, $\mathcal{V}\mathrm{ar}(\overline{u}) \subseteq \mathcal{V}\mathrm{ar}(t)$, ensuring that $t \to C(\overline{u})$ is indeed a rule, actually a rule schema since there may be many different terms $C(\overline{u})$ simplifying t.

Lemma 3.1. *$\mathcal{T}$ is a confluent and terminating rewriting system for $\leftrightarrow_T^*$.*

Proof. Since $\leftrightarrow_T^*$ extends $\sim_T$ by closure under contexts, and rewriting is closed under arbitrary context by definition, it is enough to show that $\mathcal{T}$ is a confluent and terminating rewriting system for $\sim_T$. Assume $C(\overline{u})$ simplifies t. Then $n_{\mathcal{C}}(t) > n_{\mathcal{C}}(\overline{u})$ by Lemma 2.14, ensuring termination of $\longrightarrow_{\mathcal{T}}$ since t is not headed by a constructor by definition of simplification. It follows that $\longrightarrow_{\mathcal{T}\triangleright}$ is terminating, and therefore that $(\longrightarrow_{\mathcal{T}\triangleright})_{\mathrm{mul}}$ is a well-founded relation (see for example [11] for a precise definition and the properties of the multiset extension of a relation). To show the Church-Rosser property, we interpret an equational proof as the multiset of origins of its proof steps, and reason by induction on this multiset ordered in $(\longrightarrow_{\mathcal{T}\triangleright})_{\mathrm{mul}}$. Assume $s \sim_T t$. If it contains no local peak $v \,{}_{\mathcal{T}}\!\longleftarrow u \longrightarrow_{\mathcal{T}} w$, we are done. Otherwise, standard arguments allow reducing the general case to the joinability of a critical local peak for which v, w simplify u, hence $v = C(\overline{v})$ and $w = C'(\overline{w})$. By Lemma 2.12 (ii), $C = C'$ and $\overline{v} \leftrightarrow_T^* \overline{w}$ is a collection of strictly smaller proofs. By induction hypothesis, there exists $\overline{u'}$ such that $\overline{v} \longrightarrow_{\mathcal{T}}^* \overline{u'}$ and $\overline{w} \longrightarrow_{\mathcal{T}}^* \overline{u'}$, implying joinability of the critical peak: $v \longrightarrow_{\mathcal{T}}^* C(\overline{u'}) \,{}_{\mathcal{T}}^*\!\longleftarrow w$. We have got a strictly smaller proof to which induction applies. □

3.1 Strong Normalization in CIC$^\omega$(T)

The next lemma is the basis of the coming strong normalization proof of $\longrightarrow_{\beta\iota_T}$:

Lemma 3.2. $\longrightarrow_{\beta\iota_T} \subseteq \longrightarrow_{\beta\iota\mathcal{T}}^+$ *where* $\longrightarrow_{\beta\iota\mathcal{T}} \stackrel{\text{def}}{=} \longrightarrow_\beta \cup \longrightarrow_\iota \cup \longrightarrow_{\mathcal{T}}$.

Proof. By definition of T-elimination, $\longrightarrow_{\iota_T} \subseteq (\longrightarrow_{\mathcal{T}} \cup \longrightarrow_\iota)^+$. The result follows. □

In the following, we use the short form "s, σ is SN" for s, σ is SN$(\longrightarrow_{\beta\iota\mathcal{T}})$.

By Lemma 3.2, strong normalization of $\longrightarrow_{\beta\iota\mathcal{T}}$ implies strong normalization of $\longrightarrow_{\beta\iota_T}$, our target. We therefore need to prove that all terms are SN, that is, SN for $\longrightarrow_{\beta\iota\mathcal{T}}$. The result and its proof are given below, before the several essential lemmas used in the proof.

Theorem 3.3. $\longrightarrow_{\beta\iota\mathcal{T}}$ *is strongly normalizing.*

Proof. Let s be a term. The proof that s is SN proceeds by induction on $\longrightarrow_{\beta\iota\triangleright}$. Since a term is SN iff all its reducts are SN by definition, the proof proceeds by inspecting all possible cases:

1. $s \longrightarrow_{\beta\iota} t$. Then t is SN by assumption.
2. $s \longrightarrow_{\mathcal{T}}^p t$. There are three cases according to the cap of s and the position p.
   (a) The cap of s is algebraic and $p \in \mathcal{FP}\mathrm{os}(\widehat{s})$. Since $\mathcal{T}$-rewrites are algebraic non-variable terms on both sides, rewriting in the algebraic cap of s yields a term with an algebraic cap as well whose set of aliens is a subset of the set of aliens of s by Lemma 3.4. And since $\longrightarrow_{\beta\iota\triangleright}$-reducts of s are SN, then θ_s, hence θ_t, is SN. Then, t is SN by Lemma 3.7.
   (b) The cap of s is algebraic and $p \notin \mathcal{FP}\mathrm{os}(\widehat{s})$, hence $\mathcal{A}(s)(\longrightarrow_{\mathcal{T}})_{\mathrm{mul}}\mathcal{A}(t)$ by Lemma 3.4. Since terms in $\mathcal{A}(s)$ are SN by assumption, their reducts in $\mathcal{A}(t)$ are SN, hence t is SN by Lemma 3.7 again.
   (c) The cap of s is non-algebraic, hence $p \notin \mathcal{FP}\mathrm{os}(\widehat{s})$ and $\mathcal{A}(s)(\longrightarrow_{\mathcal{T}})_{\mathrm{mul}}\mathcal{A}(t)$ by Lemma 3.4. Since terms in $\mathcal{A}(s)$ are SN by assumption, their reducts in $\mathcal{A}(t)$ are SN, hence t is SN by Lemma 3.5. □

□

We are left with the various lemmas used in the proof of Theorem 3.3

Lemma 3.4. *Let $s \longrightarrow^p_{\mathcal{T}} t$. If $p \in \mathcal{FP}\mathrm{os}(\widehat{s})$, then $\forall u \in \mathcal{A}(t)$ $(u \in \mathcal{A}(s))$. Otherwise, $\mathcal{A}(s)(\longrightarrow_{\mathcal{T}})_{\mathrm{mul}}\mathcal{A}(t)$.*

Proof. Since rewrite rules from $\mathcal{T}$ are headed by an algebraic symbol on both sides, the cap of s cannot disappear. Hence aliens in s can only be duplicated or erased by an algebraic rewriting step in the algebraic cap. On the other hand, $\mathcal{T}$-rewriting in an alien of s yields an alien of t because the non-algebaic cap of the alien will not disappear. □

We now show that a term whose aliens are SN is SN. A standard technique for such a proof is to use the cap of t, compared in $\longrightarrow_{\mathcal{T}} \cup \longrightarrow_{\beta\iota}$, as a first argument of an interpretation of the rewrite that is used to reason by induction. The difficulty here is that $\beta\iota$-rewriting does not preserve algebraic caps, since it may pop up some subterm, which may itself have an algebraic cap. The algebraic cap of the whole term get then enlarged, forbidding that sort of comparison. A similar problem occurs with non-algebraic caps. The difficulty here is that rewriting in the cap may swallow algebraic symbols heading some immediate alien in case of an $\textsc{Elim}_o(,,)$-rewrite, then enlarge that subterm again by generating algebraic terms in the right-hand side of the rule, or even collapse the entire non-algebraic cap resulting in a term which has now an algebraic cap. These various problems make the standard technique fail.

As we have seen, there are two cases, depending whether the cap is algebraic or non-algebraic. We start with the second which happens to be easier. In this case, the aforementioned difficulties are solved by using the fact that the aliens are SN: SN terms have $\beta\iota\mathcal{T}$-normal forms. Since $\longrightarrow_{\beta\iota\mathcal{T}}$ is not known yet to be confluent, each SN terms will actually have a finite multiset of normal forms, since any term has a finite number of immediate reducts. We therefore can interpret a term t whose cap is non-algebraic and aliens are SN by the set of terms obtained from t by normalizing its aliens in all possible ways. A difficulty is that identical aliens of t may become different normal forms in a term belonging to the interpretation of t. Assume now that the cap of t rewrites, necessarily with $\beta\iota$. Then, any of its normal forms will rewrite as well, because the $\beta\iota$ rules are left-linear. It would not be the case otherwise.

Lemma 3.5. *Assume that t has a non-algebraic cap and SN aliens. Then t is SN.*

Proof. Let $t{\downarrow}^{\mathrm{aliens}}_{\beta\iota\mathcal{T}}$ be the set of terms obtained from t by normalizing its aliens in all possible ways. Note that $t \longrightarrow^p_{\beta\iota\mathcal{T}} u$ with $p \neq \Lambda$ implies $u{\downarrow}^{\mathrm{aliens}}_{\beta\iota\mathcal{T}} \subseteq t{\downarrow}^{\mathrm{aliens}}_{\beta\iota\mathcal{T}}$. We prove the result by induction on the pair $\langle t{\downarrow}^{\mathrm{aliens}}_{\beta\iota\mathcal{T}}, \mathcal{A}(t)\rangle$ compared in $((\longrightarrow_{\beta\iota})_{\mathrm{mul}}, (\longrightarrow_{\beta\iota\mathcal{T}})_{\mathrm{mul}})_{\mathrm{lex}}$. This relation is well-founded, since (i) for the first component of this lexicographic comparison, $\longrightarrow_{\beta\iota}$ is terminating by Theorem 2.17 ; (ii) for the second component, aliens of t are SN by assumption ; and (iii) multiset and lexicographic extensions preserve well-foundedness (with the first multiset extension applying here to sets). By definition of the SN predicate, a term is SN iff all its reducts are SN. There are two cases:

1. If $t \longrightarrow_{\beta\iota\mathcal{T}} u$ at an alien position, then $u{\downarrow}^{\mathrm{aliens}}_{\beta\iota\mathcal{T}} \subseteq t{\downarrow}^{\mathrm{aliens}}_{\beta\iota\mathcal{T}}$, and $\mathcal{A}(t)(\longrightarrow_{\mathcal{T}})_{\mathrm{mul}}\mathcal{A}(u)$. Hence $\langle t{\downarrow}^{\mathrm{aliens}}_{\beta\iota\mathcal{T}}, \mathcal{A}(t)\rangle$ $((\longrightarrow_{\beta\iota})_{\mathrm{mul}}, (\longrightarrow_{\beta\iota\mathcal{T}})_{\mathrm{mul}})_{\mathrm{lex}}$ $\langle u{\downarrow}^{\mathrm{aliens}}_{\beta\iota\mathcal{T}}, \mathcal{A}(u)\rangle$. We therefore conclude that u is SN by induction hypothesis.
2. Let $t \longrightarrow^p_{\beta\iota} u$ at a cap position, then $t{\downarrow}^{\mathrm{aliens}}_{\beta\iota\mathcal{T}} \longrightarrow^p_{\beta\iota} u{\downarrow}^{\mathrm{aliens}}_{\beta\iota\mathcal{T}}$. Therefore, u is SN by induction hypothesis. □

□

The previous technique does not apply to terms whose aliens are SN but cap is algebraic, since rules in $\mathcal{T}$ are not left-linear. To accommodate non-left-linear rules, we need an interpretation provided by a confluent set of rules. We have one at our disposal, $\longrightarrow_{\beta\iota}$, whose termination and confluence are the result of Theorem 2.17. Some preliminary work is needed.

Lemma 3.6. *Let s be an algebraic-headed term whose aliens are SN. Assume that $s \longrightarrow^p_{\mathcal{T}} t$. If $p \in \mathcal{FP}\mathrm{os}(\widehat{s})$, then $s{\downarrow_{\beta\iota}} \longrightarrow^p_{\mathcal{T}} t{\downarrow_{\beta\iota}}$. Otherwise, $s{\downarrow_{\beta\iota}} (\longrightarrow^*_{\mathcal{T}} \longrightarrow^*_{\beta\iota})^* t{\downarrow_{\beta\iota}}$.*

Proof. By Theorem 2.17, $\beta\iota$-reductions are Church-Rosser modulo α-conversions, a property that we repeatedly use in the following without saying. Further, since α-conversion commutes with all rewrites, they can always be pushed over them all, hence be simply omitted. A last remark is that the rules in $\mathcal{T}$ have no critical pairs with the rules $\beta\iota$, hence cannot apply at the same position in s.

We compute normal forms of s and t in the following particular way: first, we normalize subterms at positions which are disjoint from p, then below p, and finally above p, using respectively, the disjoint peak property (DP), the ancestor peak property (AP) when the $\mathcal{T}$-redex is above a $\beta\iota$-redex , and the linear ancestor peak property (LAP) when the $\beta\iota$-redex is above the $\mathcal{T}$-redex. (DP), (AP) and (LAP) originate from [12], see also [14]. We upper-index the down-arrow sign by a predicate specifying which positions get normalized.

Let $s \longrightarrow^q_{\beta\iota} s' \longrightarrow^*_{\beta\iota} s{\downarrow^{\#p}_{\beta\iota}}$. By (DP), $t \longrightarrow^q_{\beta\iota} t'$ and $s' \longrightarrow^p_{\beta\iota} t'$. By induction hypothesis (w.r.t. $\beta\iota$-rewriting of s into s'), $s{\downarrow^{\#p}_{\beta\iota}} = s'{\downarrow^{\#p}_{\beta\iota}} \longrightarrow^*_{\mathcal{T}} t'{\downarrow^{\#p}_{\beta\iota}} = t{\downarrow^{\#p}_{\beta\iota}}$.

We can now assume that s, t are normal at positions disjoint from p. If $s = s{\downarrow^{>p}_{\beta\iota}}$, then $t = t{\downarrow^{>p}_{\beta\iota}}$, since the (non-algebraic headed) aliens of $t|_p$ are aliens of $s|_s$, hence $s{\downarrow^{>p}_{\beta\iota}} \longrightarrow^*_{\mathcal{T}} t{\downarrow^{>p}_{\beta\iota}}$. Otherwise, $s \longrightarrow^{q>p}_{\beta\iota} u$. By (AP), $t \longrightarrow^O_{\beta\iota} t'$, and $u \longrightarrow^Q_{\beta\iota} s' \longrightarrow^p_{\mathcal{T}} t'$, where O, Q are sets of disjoint positions bigger than p. We conclude by applying the induction hypothesis (w.r.t. $\beta\iota$-rewriting of s into s' in at least one step), that $s{\downarrow^{>p}_{\beta\iota}} = s'{\downarrow^{>p}_{\beta\iota}} \longrightarrow_{\mathcal{T}} t'{\downarrow^{>p}_{\beta\iota}} = t{\downarrow^{>p}_{\beta\iota}}$.

We can now assume further that s, t are normal at positions bigger than p. In case $p \in \mathcal{P}\mathrm{os}(\widehat{s})$, then s, t are in normal form and we are done, yielding the first conclusion of the lemma. If $p \notin \mathcal{P}\mathrm{os}(\widehat{s})$ but s is in $\beta\iota$-normal form, we are left rewriting t to its normal form $t{\downarrow_{\beta\iota}}$. This is the cause of the introduction of $\beta\eta$-steps in the lemma when $p \in \mathcal{P}\mathrm{os}(\widehat{s})$. If s is not in $\beta\iota$-normal form, let $s \longrightarrow^{q<p}_{\beta\iota} s' \longrightarrow^*_{\beta\iota}$. Because s is algebraic-headed, q is an alien position, hence p is an alien position as well. We will use here the assumption that aliens are SN to carry out the induction w.r.t. $\longrightarrow_{\beta\iota\mathcal{T}}$. Since $\beta\iota$ are left-linear rules, by (LAP), $t \longrightarrow^q_{\beta\iota} t'$, and $s' \longrightarrow^P_{\mathcal{T}}$, where P is a set of disjoint positions bigger than q. Let $s' = s'_1 \longrightarrow^P_{\mathcal{T}} s'_n = t'$. By repeatedly applying the induction hypothesis (w.r.t. $\beta\iota\mathcal{T}$-rewriting) to $s'_1, \ldots, s'_n$, we get $s{\downarrow_{\beta\iota}} = s{\downarrow^{<p}_{\beta\iota}} = s'_1{\downarrow^{<p}_{\beta\iota}} (\longrightarrow_{\mathcal{T}}^{(<p)^*} \longrightarrow^*_{\beta\iota})^* s'_2{\downarrow^{<p}_{\beta\iota}} = \ldots = (\longrightarrow_{\mathcal{T}}^{(<p)^*} \longrightarrow^*_{\beta\iota})^* s'_n{\downarrow^{<p}_{\beta\iota}} = t'{\downarrow^{<p}_{\beta\iota}} = t{\downarrow_{\beta\iota}}$, yielding the result in case $p \notin \mathcal{P}\mathrm{os}(\widehat{s})$. □

We can now prove our second lemma.

Lemma 3.7. *Assume that t has an algebraic cap and SN aliens. Then, t is SN.*

Proof. We show that immediate $\beta\iota\mathcal{T}$-reducts of t are SN by induction on the pair $\langle t{\downarrow_{\beta\iota}}, \mathcal{A}(t)\rangle$, compared in the relation $(\longrightarrow_{\mathcal{T}}, (\longrightarrow_{\beta\iota\mathcal{T}_\triangleright})_{\mathrm{mul}})_{\mathrm{lex}}$, which is well-founded since $\longrightarrow_{\mathcal{T}}$ is terminating by Lemma 3.1 and terms in $\mathcal{A}(t)$ are SN by assumption. If t is not $\beta\iota\mathcal{T}$-reducible, we are done. Otherwise, either a reduction takes place at a position in the cap, which must then be an algebraic reduction and the first argument of the pair decreases by Lemma 3.6, or it does not, and by the same lemma, the second argument of the pair decreases while the first either decreases or remains unchanged. Hence, by induction hypothesis, all $\beta\iota\mathcal{T}$-reducts of t are SN, which implies that t is SN by definition of the SN predicate. □

Using Lemma 3.2, we can now conclude:

Corollary 3.7.1. $\longrightarrow_{\beta\iota_T}$ *is strongly normalizing.*

3.2 Church-Rosser Property in CIC$^\omega$(T)

We move to the last non-trivial step, showing the Church-Rosser property of the reduction rules β, ι and T-elimination. Following [13], we show successively, in this order, the three properties which imply the main result of this section: coherence modulo $\sim_T$, termination of computation in T-congruence classes, and local confluence modulo T. Our proofs here assume that T is Presburger arithmetic for simplicity, the general case involving more notations.

Lemma 3.8 (Coherence)**.** *Assume $s \leftrightarrow_T^* s' \longrightarrow_{\beta\iota_\mathcal{T}} v$. Then, $s \longrightarrow_{\beta\iota_\mathcal{T}} w \leftrightarrow_T^* v$ for some w.*

Proof. The only delicate case is that of T-elimination, which may interact with the conversion step from t to u. Assume that $s = \textsc{Elim}_{Nat}(Q, f_\mathbf{0}, f_\mathbf{S}, t)$, $s' = \textsc{Elim}_{Nat}(Q, f_\mathbf{0}, f_\mathbf{S}, t')$, and $v = f_\mathbf{S}\ u'\ \textsc{Elim}_{Nat}(Q, f_\mathbf{0}, f_\mathbf{S}, u')$ where $C(\overline{u'})$ simplifies t'. By Lemma 2.12 (ii), there exists $C(u)$ simplifying both t and t' such that $\overline{u} \leftrightarrow_T^* \overline{u'}$. Then $w = f_\mathbf{S}\ u\ \textsc{Elim}_{Nat}(Q, f_\mathbf{0}, f_\mathbf{S}, u)$ satisfies the claim. □

The above property is a strong form of coherence, also called commutation. Together with Theorem 2.17, it readily implies, see also [11]:

Lemma 3.9 (Termination modulo T)**.** *$\leftrightarrow_T^* \longrightarrow_{\beta\iota_\mathcal{T}}$ is terminating.*

Lemma 3.10 (Local confluence modulo T)**.** *$(\forall s, t, u)\, s \longrightarrow_{\beta\iota_\mathcal{T}} t$ and $s \longrightarrow_{\beta\iota_\mathcal{T}} u$, there exist t', u' such that $t \longrightarrow_{\beta\iota_\mathcal{T}}^* t'$, $u \longrightarrow_{\beta\iota_\mathcal{T}}^* u'$, and $u' \leftrightarrow_T^* v'$.*

Proof. The proof is as usual by discussion on the position of the redexes in s, the only non-straightforward case being that of the only critical pair originating from the use of $\longrightarrow_\mathcal{T}$ to fire the $\mathcal{T}$-recursors. It is then easily shown joinable thanks to Lemma 3.1. We illustrate the computation in case of Presburger arithmetic. Consider the rule $\textsc{Elim}_{Nat}(Q, f_\mathbf{0}, f_\mathbf{S}, t) \rightarrow_{\iota_\mathcal{T}} f_\mathbf{S}\ u\ \textsc{Elim}_{Nat}(Q, f_\mathbf{0}, f_\mathbf{S}, u)$ if $S(u)$ simplifies t. Assume $S(u)$ and $S(v)$ both simplify t, hence $S(u) \leftrightarrow_T^* S(v)$, yielding the critical pair $\langle f_\mathbf{S}\ u\ \textsc{Elim}_{Nat}(Q, f_\mathbf{0}, f_\mathbf{S}, u),\ f_\mathbf{S}\ v\ \textsc{Elim}_{Nat}(Q, f_\mathbf{0}, f_\mathbf{S}, v)\rangle$. By lemma 3.1, $S(u)$ and $S(v)$ (hence u and v) have the same $\mathcal{T}$-normal form, implying $u \leftrightarrow_T^* v$. Hence, by Definition 2.9: $f_\mathbf{S}\ u\ \textsc{Elim}_{Nat}(Q, f_\mathbf{0}, f_\mathbf{S}, u) \leftrightarrow_T^* f_\mathbf{S}\ v\ \textsc{Elim}_{Nat}(Q, f_\mathbf{0}, f_\mathbf{S}, v)$. □

Theorem 3.11 (Church-Rosser property of $\longrightarrow_{\beta\iota_\mathcal{T}}$ modulo T)**.** *$(\forall u, v)\, u \simeq v$, there exists u', v' such that $u \longrightarrow_{\beta\iota_\mathcal{T}}^* u'$, $v \longrightarrow_{\beta\iota_\mathcal{T}}^* v'$, and $u' \leftrightarrow_T^* v'$.*

Proof. Follows from [13] by using Lemmas 3.9, 3.8 and 3.10. □

3.3 Decidability of Type Checking

We can now easily decide convertibility by rewriting to normal forms: taking two arbitrary typable terms u, v, then, by the Church-Rosser property of $\longrightarrow_{\beta\iota_\mathcal{T}}$, $u \simeq v$ iff $u{\downarrow_{\beta\iota_\mathcal{T}}} \leftrightarrow_T^* v{\downarrow_{\beta\iota_\mathcal{T}}}$. The technique for showing decidability of type checking in CIC$^\omega$(T) is then standard: by incorporating conversion to the application rule, we get an equivalent syntax oriented type checker whose decidability reduces to the decidability of conversion. See for example [6].

4 Programming with CoqMT

From its initial version, CoqMT eases the development of libraries with dependent data-types, as exemplified in the dependent vector case study of Figure 3, where statements and proofs are canonically lifted from their non-dependent counter-parts. (The full development can be

```
Variable T : Type.

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

Fixpoint append n1 n2
  (xs1 : dlist n1) (xs2 : dlist n2)
:=
  match xs1 in dlist n1
    return dlist (n1 + n2) with
  | nil ⇒ xs2
  | cons n1 x xs1 ⇒
      cons x (append xs1 xs2)
  end.
```

```
Infix "@" := append.

Lemma appA n1 n2 n3
  (xs1 : dlist n1)
  (xs2 : dlist n2)
  (xs3 : dlist n3):
  (xs @ ys) @ zs = xs @ (ys @ zs).
Proof.
  induction xs as [|n x xs IH].
    trivial.
  intros ys zs; simpl.
  rewrite IH; reflexivity.
Qed.
```

Figure 3: Reasoning about Dependent Vectors in CoqMT

obtained from the CoqMT webpage. This development does not currently include the new version of CoqMT with the reduction modulo T studied here. This is work in progress which requires a non-trivial change in the reduction machinery of Coq.) Because of its former lack of strong recursion, CoqMT failed to ease the development of libraries relying on dependent data-types defined by (strong)-induction over an integer parameter.

A standard example of strong recursors computing over natural numbers is that of an iterated structure. The type of multinomials over a fixed number of indeterminates, when built by iterating the 1-indeterminate case, is such a structure. Assuming we have a type constructor `poly : Type → Type` such that `poly K` stands for the type of polynomials in 1 indeterminate over `K`, we can construct the type `mpoly K n`, of multinomials over `n` indeterminates over `K` as:

```
Fixpoint mpoly (K : ring) (n : nat) : Type :=
  match n with 0 ⇒ K | S p ⇒ poly (mpoly K p).
```

With this kind of structure, as in the vector case, it is common that some arithmetical reasoning about the indexes (here, the number of indeterminates) occurs; and, the more indexes we identify, the less type casts has to be used. Indeed, any type cast that involves identifying two different indexes equal in $\mathcal{T}$ can be removed in CoqMT. For instance, one may want to identify `mpoly K (n+1+p)` with `mpoly K (p+n+1)`. In plain Coq, or in previous versions of CoqMT, these types were considered unequal, either because `n+1+p` and `p+n+1` were not identified (in Coq), or because their identification was not used by the (strong) recursor rule (in CoqMT). This case can be exemplified with the multinomials evaluation function `meval`:

```
Definition meval (K : ring) (n : nat) : mpoly K n → dlist K n → K.
```

which has, for example, the following natural property:

```
Definition mevalS K n (p : mpoly K (S n)) (x : dlist K (S n)) :
  meval p x = meval (eval p (dhead x)) (dtail x)
```

where `dhead` and `dtail` are the functions returning resp. the head and tail of a non-empty list (i.e. of size greater than 0), `eval` is the evaluation function for univariate polynomials (i.e. for objects of type `poly`). Note that by definition of `mpoly`, an object of type `mpoly K (S n)` is identified, by ι-reduction (using the recursor rules), with an object of type `poly(mpoly K n)`.

```
Definition vhead (T : Type) n : dlist T (S n) → T.
Definition vtail (T : Type) n : dlist T (S n) → dlist T n.
Definition eval (K : ring) : poly K → K → K.
```

In the new version of CoqMT justified here, not only `mpoly K (n+1+p)` and `mpoly K (p+n+1)` are identified, but because `n+1+p` simplifies `S (n+p)`, they compute to `poly (mpoly K (n+p))`, providing some canonical form of our initial type which highlights that `poly` is iterated at least once. This would allow, for instance, to easily use properties like `mevalS` on multivariate polynomials without relying on unnecessary type casts. For instance, we noticed that such needs arise quite naturally in the proof of the symmetric polynomials fundamental lemma, where all type casts occurring in the proof could be effectively removed in CoqMT.[1] We also suspect that the development of any library that does non-trivial calculations of multivariate polynomials would be eased by the use of CoqMT. For example, a formal library for the theory of Gröbner bases would be a good target for CoqMT.

Formally, given that the conversion can be decided by the syntactic equality of expressions in normal form for the confluent T reduction, no type cast is needed provided the theory T comes with all the vocabulary used in the formalization. The formalization could of course need more vocabulary. For instance, multiplication over natural numbers would be needed as soon as we formalize multiplication over polynomials. Then, we would need to include multiplication in the theory T in order to avoid new type casts. Enriching the theory T can of course continue as long as decidability is still satisfied, which pinpoints precisely when type casts become necessary.

5 Conclusion

Coq Modulo Theory (CoqMT) implements a type theory in which conversion can be defined extensionally at small (first-order) inductive types. CoqMT is much closer to the developer's intuition because types depending on expressions which are extensionally equal become identifiable. In case of Presburger arithmetic, for example, $0 + x$ and x are extensionally equal, but not intensionally equal in case the function symbol $+$ is defined by induction on its second argument. This allows for a liberal type checker when dependent types come into play.

In this paper, we have shown that type checking is decidable in CoqMT, even in presence of strong elimination modulo a (decidable) theory. The proof is quite interesting, in that it relies on the strong normalization of the calculus with strong elimination to show strong normalization of the calculus with strong elimination modulo a theory. This syntactic proof reveals a new mechanism that could be added to Barras semantic model construction used to prove strong normalization and consistency of CoqMT with strong elimination. This would enhance its expressivity in case of interactions, like here, between different syntactic constructs which depend on each other, here the theory T used both in the conversion typing rule, and in the weak and strong elimination rules.

Since (the formal model of) Coq can be faithfully embedded in (the formal model of) Dedukti, one can ask whether CoqMT can also be faithfully embedded in Dedukti [2]. Since Dedukti does not have a primitive extensional equality, this implies that the embedding should use explicit casts. Eliminating these casts would of course be possible by extending Dedukti capabilities with a decidable extensional conversion as we did for Coq. This would of course impact the meta-theory of Dedukti. We can expect that the strong normalization can be carried out as we did here. But in Dedukti, another important difficulty is the confluence

[1] A formalization of this result, done in plain Coq at that time, can be found at https://github.com/math-comp/multinomials

property because of the presence of rewrite rules which may have critical pairs. This implies that the confluence proof has to be carried out on untyped terms [3]. Adding an extensional equality as in CoqMT would be a important challenge for the confluence proof that has not be considered yet for Dedukti.

References

[1] Thorsten Altenkirch. Proving Strong Normalization of CC by Modifying Realizability Semantics. In Henk Barendregt and Tobias Nipkow, editors, *TYPES*, volume 806 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 1993.

[2] Ali Assaf. A calculus of constructions with explicit subtyping. In Hugo Herbelin, Pierre Letouzey, and Matthieu Sozeau, editors, *20th International Conference on Types for Proofs and Programs, TYPES 2014, May 12-15, 2014, Paris, France*, volume 39 of *LIPIcs*, pages 27–46. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[3] Ali Assaf, Gilles Dowek, Jean-Pierre Jouannaud, and Jiaxiang Liu. Untyped confluence in dependent type theory, 2017. submitted.

[4] Bruno Barras. Semantical Investigations in Intuitionistic Set Theory and Type Theories with Inductive Families. Habilitation thesis, http://www.lix.polytechnique.fr/ barras/habilitation/.

[5] Bruno Barras. Sets in Coq, Coq in Sets. *J. Formalized Reasoning*, 3(1):29–48, 2010.

[6] Bruno Barras, Jean-Pierre Jouannaud, Pierre-Yves Strub, and Qian Wang. CoQMTU: A Higher-Order Type Theory with a Predicative Hierarchy of Universes Parametrized by a Decidable First-Order Theory. In *LICS*, pages 143–151. IEEE Computer Society, 2011.

[7] Frédéric Blanqui. Inductive Types in the Calculus of Algebraic Constructions. In Martin Hofmann, editor, *TLCA*, volume 2701 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2003.

[8] Frédéric Blanqui, Jean-Pierre Jouannaud, and Pierre-Yves Strub. From Formal Proofs to Mathematical Proofs: A Safe, Incremental Way for Building in First-order Decision Procedures. In Giorgio Ausiello, Juhani Karhumäki, Giancarlo Mauri, and C.-H. Luke Ong, editors, *IFIP TCS*, volume 273 of *IFIP*, pages 349–365. Springer, 2008.

[9] Adel Bouhoula. SPIKE: a system for sufficient completeness and parameterized inductive proofs. In Alan Bundy, editor, *Automated Deduction - CADE-12, 12th International Conference on Automated Deduction, Nancy, France, June 26 - July 1, 1994, Proceedings*, volume 814 of *Lecture Notes in Computer Science*, pages 836–840. Springer, 1994.

[10] Thierry Coquand and Gérard P. Huet. The Calculus of Constructions. *Inf. Comput.*, 76(2/3):95–120, 1988.

[11] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite Systems. In North Holland, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 243–320. J. van Leuven, 1990.

[12] G. Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, October 1980.

[13] Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a Set of Rules Modulo a Set of Equations. *SIAM J. Comput.*, 15(4):1155–1194, 1986.

[14] Jean-Pierre Jouannaud and Jianqi Li. Church-Rosser properties of normal rewriting. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) - 21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPIcs*, pages 350–365. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[15] Zhaohui Luo. ECC, an Extended Calculus of Constructions. In *LICS*, pages 386–395. IEEE Computer Society, 1989.

[16] Per Martin-Löf. Constructive mathematics and computer programming. In H. Pfeiffer L.J. Cohen, J. Los and K.-P. Podewski, editors, *in Logic, methodology and philosophy of science VI, Proceedings*

of the 1979 international congress at Hannover, Germany, page 153–175. North- Holland, 1982.

[17] Nicolas Oury. Extensionality in the Calculus of Constructions. In Joe Hurd and Thomas F. Melham, editors, *TPHOLs*, volume 3603 of *Lecture Notes in Computer Science*, pages 278–293. Springer, 2005.

[18] Christine Paulin-Mohring. Inductive Definitions in the system Coq - Rules and Properties. In Marc Bezem and Jan Friso Groote, editors, *TLCA*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 1993.

[19] Mark-Oliver Stehr. The Open Calculus of Constructions (Part I): An Equational Type Theory with Dependent Types for Programming, Specification, and Interactive Theorem Proving. *Fundam. Inform.*, 68(1-2):131–174, 2005.

[20] Pierre-Yves Strub. Coq Modulo Theory. In Anuj Dawar and Helmut Veith, editors, *CSL*, volume 6247 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2010.

[21] The Coq Development Team. The Coq Proof Assistant, Reference Manual, Version 8.4. Technical report, INRIA, Roquencourt, France, 2012.

[22] Qian Wang and Bruno Barras. Semantics of Intensional Type Theory extended with Decidable Equational Theories. In Simona Ronchi Della Rocca, editor, *CSL*, volume 23 of *LIPIcs*, pages 653–667. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[23] Benjamin Werner. Sets in Types, Types in Sets. In Martín Abadi and Takayasu Ito, editors, *TACS*, volume 1281 of *Lecture Notes in Computer Science*, pages 530–346. Springer, 1997.