Inductive types in the Calculus of Algebraic Constructions

Frédéric Blanqui

Inductive types in the Calculus of Algebraic Constructions

Frédéric Blanqui

Laboratoire d'Informatique de l'École Polytechnique
91128 Palaiseau Cedex, France
blanqui@lix.polytechnique.fr

Abstract. In a previous work, we proved that almost all of the Calculus of Inductive Constructions (CIC), the basis of the proof assistant Coq, can be seen as a Calculus of Algebraic Constructions (CAC), an extension of the Calculus of Constructions with functions and predicates defined by higher-order rewrite rules. In this paper, we prove that CIC as a whole can be seen as a CAC, and that it can be extended with non-strictly positive types and inductive-recursive types together with non-free constructors and pattern-matching on defined symbols.

1 Introduction

There has been different proposals for defining inductive types and functions in typed systems. In Girard's polymorphic λ-calculus or in the Calculus of Constructions (CC) [9], data types and functions can be formalized by using impredicative encodings, difficult to use in practice, and computations are done by β-reduction only. In Martin-Löf's type theory or in the Calculus of Inductive Constructions (CIC) [10], inductive types and their induction principles are first-class objects, functions can be defined by induction and computations are done by ι-reduction. For instance, for the type nat of natural numbers, the recursor $rec : (P : nat \Rightarrow \star)(u : P0)(v : (n : nat)Pn \Rightarrow P(sn))(n : nat)Pn$ is defined by the following ι-rules:

$$\begin{array}{rcl} rec\ P\ u\ v\ 0 & \rightarrow_\iota & u \\ rec\ P\ u\ v\ (s\ n) & \rightarrow_\iota & v\ n\ (rec\ P\ u\ v\ n) \end{array}$$

Finally, in the algebraic setting [11], functions are defined by using rewrite rules and computations are done by applying these rules. Since both β-reduction and ι-reduction are particular cases of higher-order rewriting [16], proposals soon appeared for integrating all these approaches. Starting with [15,2], this objective culminated with [4,5,6] in which almost all of CIC can be seen as a Calculus of Algebraic Constructions (CAC), an extension of CC with functions and predicates defined by higher-order rewrite rules. In this paper, we go one step further in this direction and capture all previous proposals, and much more.

Let us see the two examples of recursors that are allowed in CIC but not in CAC [20]. The first example is a third-order definition of finite sets of natural numbers (represented as predicates over nat):

$$
\begin{aligned}
\mathit{fin} &: (\mathit{nat} \Rightarrow \star) \Rightarrow \star \\
\mathit{femp} &: \mathit{fin}\,\emptyset \\
\mathit{fadd} &: (x : \mathit{nat})(p : \mathit{nat} \Rightarrow \star)\mathit{fin}\,p \Rightarrow \mathit{fin}(\mathrm{add}\ x\ p) \\
\mathit{rec} &: (Q : (\mathit{nat} \Rightarrow \star) \Rightarrow \star)Q\emptyset \\
&\quad \Rightarrow ((x : \mathit{nat})(p : \mathit{nat} \Rightarrow \star)\mathit{fin}\,p \Rightarrow Qp \Rightarrow Q(\mathrm{add}\ x\ p)) \\
&\quad \Rightarrow (p : \mathit{nat} \Rightarrow \star)\mathit{fin}\,p \Rightarrow Qp
\end{aligned}
$$

where $\emptyset = [y : \mathit{nat}]\bot$ represents the empty set, $\mathrm{add}\ x\ p = [y : \mathit{nat}]y = x \vee (p\ y)$ represents the set $\{x\} \cup p$, and the weak recursor *rec* (recursor for defining objects) is defined by the rules:

$$
\begin{array}{rcl}
\mathit{rec}\ Q\ u\ v\ p'\ \mathit{femp} & \rightarrow & u \\
\mathit{rec}\ Q\ u\ v\ p'\ (\mathit{fadd}\ x\ p\ h) & \rightarrow & v\ x\ p\ h\ (\mathit{rec}\ Q\ u\ v\ p\ h)
\end{array}
$$

The problem comes from the fact that, in $\mathit{fin}(\mathrm{add}\ x\ p)$, the output type of *fadd*, the predicate p is not a parameter of *fin*.[1] This can be generalized to any big/impredicative dependent type, that is, to any type having a constructor with a predicate argument which is not a parameter. Formally, if $C : (\boldsymbol{z} : \boldsymbol{V})\star$ is a type and $c : (\boldsymbol{x} : \boldsymbol{T})C\boldsymbol{v}$ is a constructor of C then, for all predicate variable x occurring in some T_j, there must be some argument $v_{\iota_x} = x$, a condition called (I6) in [5].

The second example is John Major's equality which is intended to equal terms of different types [18]:

$$
\begin{aligned}
\mathit{JMeq} &: (A : \star)A \Rightarrow (B : \star)B \Rightarrow \star \\
\mathit{refl} &: (A : \star)(x : A)(\mathit{JMeq}\ A\ x\ A\ x) \\
\mathit{rec} &: (A : \star)(x : A)(P : (B : \star)B \Rightarrow \star)(P\ A\ x) \\
&\quad \Rightarrow (B : \star)(y : B)(\mathit{JMeq}\ A\ x\ B\ y) \Rightarrow (P\ B\ y)
\end{aligned}
$$

where *rec* is defined by the rule:

$$
\mathit{rec}\ C\ x\ P\ h\ C\ x\ (\mathit{refl}\ C\ x) \quad \rightarrow \quad h
$$

Here, the problem comes from the fact that the argument for B is equal to the argument for A. This can be generalized to any polymorphic type having a constructor with two equal type parameters. From a rewriting point of view, this is like having pattern-matching or non-linearities on predicate arguments, which is known to create inconsistencies in some cases [14]. Formally, a rule $f\boldsymbol{l} \rightarrow r$ with $f : (\boldsymbol{x} : \boldsymbol{T})U$ is *safe* if, for all predicate argument x_i, l_i is a variable and, if x_i and x_j are two distinct predicate arguments, then $l_i \neq l_j$. An inductive type is *safe* if the corresponding ι-rules are safe.

By using what is called in Matthes' terminology [17] an *elimination-based* interpretation instead of the *introduction-based* interpretation that we used in [5], we prove that recursors for types like *fin* or *JMeq* can be accepted, hence that CAC essentially subsumes CIC. In addition, we prove that it can be extended to non-strictly positive types (Section 7) and to inductive-recursive types [12] (Section 8).

[1] This is also the reason why the corresponding strong recursor, that is, the recursor for defining types or predicates, is not allowed in CIC (p could be "bigger" than *fin*).

2 The Calculus of Algebraic Constructions (CAC)

We assume the reader familiar with typed λ-calculi [3] and rewriting [11]. The Calculus of Algebraic Constructions (CAC) [5] simply extends CC by considering a set $\mathcal{F}$ of *symbols*, equipped with a total quasi-ordering $\geq$ (precedence) whose strict part is well-founded, and a set $\mathcal{R}$ of *rewrite rules*. The terms of CAC are:

$$t ::= s \mid x \mid f \mid [x:t]u \mid tu \mid (x:t)u$$

where $s \in \mathcal{S} = \{\star, \Box\}$ is a *sort*, $x \in \mathcal{X}$ a *variable*, $f \in \mathcal{F}$, $[x:t]u$ an *abstraction*, tu an *application*, and $(x:t)u$ a *dependent product*, written $t \Rightarrow u$ if x does not freely occur in u. We denote by $\mathrm{FV}(t)$ the set of free variables of t, by $\mathrm{Pos}(t)$ the set of Dewey's positions of t, and by $\mathrm{dom}(\theta)$ the *domain* of a substitution θ.

The sort $\star$ denotes the universe of types and propositions, and the sort $\Box$ denotes the universe of predicate types (also called *kinds*). For instance, the type nat of natural numbers is of type $\star$, $\star$ itself is of type $\Box$ and $nat \Rightarrow \star$, the type of predicates over nat, is of type $\Box$.

Every symbol f is equipped with a sort s_f, an *arity* α_f and a type τ_f which may be any closed term of the form $(\boldsymbol{x}:\boldsymbol{T})U$ with $|\boldsymbol{x}| = \alpha_f$ ($|\boldsymbol{x}|$ is the length of $\boldsymbol{x}$). We denote by Γ_f the environment $\boldsymbol{x}:\boldsymbol{T}$. The terms only built from variables and applications of the form $f\boldsymbol{t}$ with $|\boldsymbol{t}| = \alpha_f$ are called *algebraic*.

A rule for typing symbols is added to the typing rules of CC:

$$(\mathrm{symb}) \qquad \frac{\vdash \tau_f : s_f}{\vdash f : \tau_f}$$

A *rewrite rule* is a pair $l \to r$ such that (1) l is algebraic, (2) l is not a variable, and (3) $\mathrm{FV}(r) \subseteq \mathrm{FV}(l)$. A symbol f with no rule of the form $f\boldsymbol{l} \to r$ is *constant*, otherwise it is (partially) *defined*. We also assume that, in every rule $f\boldsymbol{l} \to r$, the symbols occurring in r are smaller than or equivalent to f.

Finally, in CAC, $\beta\mathcal{R}$-equivalent types are identified. More precisely, in the type conversion rule of CC, $\downarrow_\beta$ is replaced by $\downarrow_{\beta\mathcal{R}}$:

$$(\mathrm{conv}) \qquad \frac{\Gamma \vdash t : T \quad T \downarrow_{\beta\mathcal{R}} T' \quad \Gamma \vdash T' : s}{\Gamma \vdash t : T'}$$

where $u \downarrow_{\beta\mathcal{R}} v$ iff there exists a term w such that $u \to^*_{\beta\mathcal{R}} w$ and $v \to^*_{\beta\mathcal{R}} w$, $\to^*_{\beta\mathcal{R}}$ being the reflexive and transitive closure of $\to = \to_\beta \cup \to_\mathcal{R}$. This rule means that any term t of type T in the environment Γ is also of type T' if T and T' have a common reduct (and T' is of type some sort s). For instance, if t is a proof of $P(2+2)$ then t is also a proof of $P(4)$ if $\mathcal{R}$ contains the following rules:

$$\begin{array}{rcl} x + 0 & \to & x \\ x + (s\ y) & \to & s\ (x+y) \end{array}$$

This allows to decrease the size of proofs by an important factor, and to increase the automation as well. **All over the paper, we assume that $\to$ is confluent**.

A substitution θ *preserves typing from Γ to Δ*, written $\theta : \Gamma \rightsquigarrow \Delta$, if, for all $x \in \mathrm{dom}(\Gamma)$, $\Delta \vdash x\theta : x\Gamma\theta$, where $x\Gamma$ is the type associated to x in Γ. Type-preserving substitutions enjoy the following important property: if $\Gamma \vdash t : T$ and $\theta : \Gamma \rightsquigarrow \Delta$ then $\Delta \vdash t\theta : T\theta$.

For ensuring the *subject reduction* property (preservation of typing under reduction), every rule $f\boldsymbol{l} \to r$ is equipped with an environment Γ and a substitution ρ such that, if $f : (\boldsymbol{x} : \boldsymbol{T})U$ and $\gamma = \{\boldsymbol{x} \mapsto \boldsymbol{l}\}$, then $\Gamma \vdash f\boldsymbol{l}\rho : U\gamma\rho$ and $\Gamma \vdash r : U\gamma\rho$. The substitution ρ allows to eliminate non-linearities due to typing. For instance, the concatenation on polymorphic lists (type $list : \star \Rightarrow \star$ with constructors $nil : (A : \star)listA$ and $cons : (A : \star)A \Rightarrow listA \Rightarrow listA$) of type $(A : \star)listA \Rightarrow listA \Rightarrow listA$ can be defined by:

$$\begin{array}{rcl} app\ A\ (nil\ A')\ l' & \to & l' \\ app\ A\ (cons\ A'\ x\ l)\ l' & \to & cons\ A\ x\ (app\ A\ x\ l\ l') \\ app\ A\ (app\ A'\ l\ l')\ l'' & \to & app\ A\ l\ (app\ A\ l'\ l'') \end{array}$$

with $\Gamma = A : \star, x : A, l : listA, l' : listA$ and $\rho = \{A' \mapsto A\}$. For instance, $app\ A\ (nil\ A')$ is not typable in Γ (since $A' \notin \mathrm{dom}(\Gamma)$) but becomes typable if we apply ρ. This does not matter since, if an instance $app\ A\sigma\ (nil\ A'\sigma)$ is typable then $A\sigma$ is convertible to $A'\sigma$. Eliminating non-linearities makes rewriting more efficient and the proof of confluence easier.

3 Strong normalization

Typed λ-calculi are generally proved strongly normalizing by using Tait and Girard's technique of *reducibility candidates* [13]. The idea of Tait, later extended by Girard to the polymorphic λ-calculus, is to strengthen the induction hypothesis. Instead of proving that every term is strongly normalizable (set $\mathcal{SN}$), one associates to every type T a set $[\![T]\!] \subseteq \mathcal{SN}$, the *interpretation* of T, and proves that every term t of type T is *computable*, *i.e.* belongs to $[\![T]\!]$. Hereafter, we follow the proof given in [7] which greatly simplifies the one given in [5].

Definition 1 (Reducibility candidates) A term t is *neutral* if it is not an abstraction, not of the form $c\boldsymbol{t}$ with $c : (\boldsymbol{y} : \boldsymbol{U})C\boldsymbol{v}$ and C constant, nor of the form $f\boldsymbol{t}$ with f defined and $|\boldsymbol{t}| < \alpha_f$. We inductively define the complete lattice $\mathcal{R}_t$ of the interpretations for the terms of type t, the ordering $\leq_t$ on $\mathcal{R}_t$, and the greatest element $\top_t \in \mathcal{R}_t$ as follows.

- $\mathcal{R}_t = \{\emptyset\}$, $\leq_t = \subseteq$ and $\top_t = \emptyset$ if $t \neq \Box$ and $\Gamma \not\vdash t : \Box$.
- $\mathcal{R}_s$ is the set of subsets $R \subseteq \mathcal{T}$ such that:
  (R1) $R \subseteq \mathcal{SN}$ (strong normalization).
  (R2) If $t \in R$ then $\to(t) = \{t' \mid t \to t'\} \subseteq R$ (stability by reduction).
  (R3) If t is neutral and $\to(t) \subseteq R$ then $t \in R$ (neutral terms).
  Furthermore, $\leq_s = \subseteq$ and $\top_s = \mathcal{SN}$.

– $\mathcal{R}_{(x:U)K}$ is the set of functions R from $\mathcal{T} \times \mathcal{R}_U$ to $\mathcal{R}_K$ such that $R(u, S) = R(u', S)$ whenever $u \to u'$, $R \leq_{(x:U)K} R'$ iff, for all $(u, S) \in \mathcal{T} \times \mathcal{R}_U$, $R(u, S) \leq_K R'(u, S)$, and $\top_{(x:U)K}(u, S) = \top_K$.

Note that $\mathcal{R}_t = \mathcal{R}_{t'}$ whenever $t \to t'$ and that, for all $R \in \mathcal{R}_s$, $\mathcal{X} \subseteq R$.

Definition 2 (Interpretation schema) A *candidate assignment* is a function ξ from $\mathcal{X}$ to $\bigcup\{\mathcal{R}_t \mid t \in \mathcal{T}\}$. An assignment ξ *validates* an environment Γ, written $\xi \models \Gamma$, if, for all $x \in \mathrm{dom}(\Gamma)$, $x\xi \in \mathcal{R}_{x\Gamma}$. An *interpretation* for a symbol f is an element of $\mathcal{R}_{\tau_f}$. An *interpretation* for a set $\mathcal{G}$ of symbols is a function which, to each symbol $g \in \mathcal{G}$, associates an interpretation for g.

The *interpretation* of t w.r.t. a candidate assignment ξ, an interpretation I for $\mathcal{F}$ and a substitution θ, is defined by induction on t as follows.

- $[\![t]\!]^I_{\xi,\theta} = \top_t$ if t is an object or a sort,
- $[\![x]\!]^I_{\xi,\theta} = x\xi$,
- $[\![f]\!]^I_{\xi,\theta} = I_f$,
- $[\![(x : U)V]\!]^I_{\xi,\theta} = \{t \in \mathcal{T} \mid \forall u \in [\![U]\!]^I_{\xi,\theta}, \forall S \in \mathcal{R}_U, tu \in [\![V]\!]^I_{\xi^S_x,\theta^u_x}\}$,
- $[\![[x : U]v]\!]^I_{\xi,\theta}(u, S) = [\![v]\!]^I_{\xi^S_x,\theta^u_x}$,
- $[\![tu]\!]^I_{\xi,\theta} = [\![t]\!]^I_{\xi,\theta}(u\theta, [\![u]\!]^I_{\xi,\theta})$,

where $\xi^S_x = \xi \cup \{x \mapsto S\}$ and $\theta^u_x = \theta \cup \{x \mapsto u\}$. A substitution θ is *adapted* to a Γ-assignment ξ if $\mathrm{dom}(\theta) \subseteq \mathrm{dom}(\Gamma)$ and, for all $x \in \mathrm{dom}(\theta)$, $x\theta \in [\![x\Gamma]\!]^I_{\xi,\theta}$. A pair (ξ, θ) is Γ*-valid*, written $\xi, \theta \models \Gamma$, if $\xi \models \Gamma$ and θ is adapted to ξ.

Note that $[\![t]\!]^I_{\xi,\theta} = [\![t]\!]^{I'}_{\xi',\theta'}$ whenever ξ and ξ' agree on the predicate variables free in t, θ and θ' agree on the variables free in t, and I and I' agree on the symbols occurring in t. The difficult point is then to define an interpretation for predicate symbols and to prove that every symbol f is computable (*i.e.* $f \in [\![\tau_f]\!]$).

Following previous works on inductive types [19,23], the interpretation of a constant predicate symbol C is defined as the least fixpoint of a monotone function $I \mapsto \varphi^I_C$ on the complete lattice $\mathcal{R}_{\tau_C}$. Following Matthes [17], there is essentially two possible definitions that we illustrate by the case of *nat*. The *introduction-based* definition:

$$\varphi^I_{nat} = \{t \in \mathcal{SN} \mid t \to^* su \Rightarrow u \in I\}$$

and the *elimination-based* definition:

$$\varphi^I_{nat} = \{t \in \mathcal{T} \mid \forall (\xi, \theta)\ \Gamma\text{-valid},\ rec\ P\theta\ u\theta\ v\theta\ t \in [\![Pn]\!]^I_{\xi,\theta^t_n}\}$$

where $\Gamma = P : nat \Rightarrow \star, u : P0, v : (n : nat)Pn \Rightarrow P(sn)$. In both cases, the monotonicity of φ_{nat} is ensured by the fact that *nat* occurs only positively[2] in

[2] X occurs positively in $Y \Rightarrow X$ and negatively in $X \Rightarrow Y$. In Section 8, we give an extended definition of positivity for dealing with inductive-recursive types [12].

the types of the arguments of its constructors, a common condition for inductive types.[3].

In [5], we used the introduction-based approach since this allows us to have non-free constructors and pattern-matching on defined symbols, which is forbidden in CIC and does not seem possible with the elimination-based approach. Indeed, in CAC, it is possible to formalize the type *int* of integers by taking the symbols $0 : int$, $s : int \Rightarrow int$ and $p : int \Rightarrow int$, together with the rules:

$$\begin{array}{rcl} s\ (p\ x) & \rightarrow & x \\ p\ (s\ x) & \rightarrow & x \end{array}$$

It is also possible to have the following rule on natural numbers:

$$x \times (y + z) \quad \rightarrow \quad (x \times y) + (x \times z)$$

To this end, we extended the notion of constructor by considering as *constructor* any symbol c whose output type is a constant predicate symbol C (perhaps applied to some arguments). Then, the arguments of c that can be used to define the result of a function are restricted to the arguments, called *accessible*, in the type of which C occurs only positively. We denote by $\mathrm{Acc}(c)$ the set of accessible arguments of c. For instance, x is accessible in sx since nat occurs only positively in the type of x. But, we also have x and y accessible in $x + y$ since nat occurs only positively in the types of x and y. So, $+$ can be seen as a constructor too.

With this approach, we can safely take:

$$\varphi^I_{nat} = \{t \in \mathcal{SN} \mid \forall f, t \rightarrow^* f\boldsymbol{u} \Rightarrow \forall j \in \mathrm{Acc}(f), u_j \in [\![U_j]\!]^I_{\xi,\theta}\}$$

where $f : (\boldsymbol{y} : \boldsymbol{U})C\boldsymbol{v}$ and $\theta = \{\boldsymbol{y} \mapsto \boldsymbol{u}\}$, whenever an appropriate assignment ξ for the predicate variables of U_j can be defined, which is possible only if the condition (I6) is satisfied (see the type *fin* in Section 1).

4 Extended recursors

As we introduced an extended notion of constructor for dealing with the introduction-based method, we now introduce an extended notion of recursor for dealing with the elimination-based method.

Definition 1 (Extended recursors). *A* pre-recursor *for a constant predicate symbol $C : (\boldsymbol{z} : \boldsymbol{V})\star$ is any symbol f such that:*

- *the type of f is of the form*[4] $(\boldsymbol{z} : \boldsymbol{V})(z : C\boldsymbol{z})W$*,*

[3] Mendler proved that recursors for negative types are not normalizing [19]. Take for instance an inductive type C with a constructor $c : (C \rightarrow nat) \rightarrow C$. Assume now that we have $p : C \rightarrow (C \rightarrow nat)$ defined by the rule $p(cx) \rightarrow x$ (case analysis). Then, by taking $\omega = [x : C](px)x$, we get $\omega(c\omega) \rightarrow_\beta p(c\omega)(c\omega) \rightarrow \omega(c\omega) \rightarrow_\beta \ldots$

[4] Our examples may not always fit in this form but since, in an environment, two types that do not depend on each other can be permuted, this does not matter.

- *every rule defining f is of the form $f\boldsymbol{z}t\boldsymbol{u} \rightarrow r$ with* $\mathrm{FV}(r) \cap \{\boldsymbol{z}\} = \emptyset$,
- $f\boldsymbol{v}t\boldsymbol{u}$ *is head-reducible only if t is constructor-headed.*

A pre-recursor f is a recursor *if it satisfies the following* positivity conditions:[5]

- *no constant predicate $D > C$ or defined predicate F occurs in W,*
- *every constant predicate $D \simeq C$ occurs only positively in W.*

A recursor of sort $\star$ (resp. $\Box$) is weak *(resp.* strong*). Finally, we assume that every type C has a set $\mathcal{R}ec(C)$ (possibly empty) of recursors.*

For the types C whose set of recursors $\mathcal{R}ec(C)$ is not empty, we define the interpretation of C with the elimination-based method as follows. For the other types, we keep the introduction-based method.

Definition 2 (Interpretation of inductive types). *If every t_i has a normal form t_i^* then $\varphi_C^I(\boldsymbol{t}, \boldsymbol{S})$ is the set of terms t such that, for all $f \in \mathcal{R}ec(C)$ of type $(\boldsymbol{z} : \boldsymbol{V})(z : C\boldsymbol{z})(\boldsymbol{y} : \boldsymbol{U})V$, $\boldsymbol{y}\xi$ and $\boldsymbol{y}\theta$, if $\xi_{\boldsymbol{z}}^{\boldsymbol{S}}, \theta_{\boldsymbol{z}z}^{\boldsymbol{t}t} \models \boldsymbol{y} : \boldsymbol{U}$ then $f\boldsymbol{t}^*t\boldsymbol{y}\theta \in [\![V]\!]^I_{\xi_{\boldsymbol{z}}^{\boldsymbol{S}}, \theta_{\boldsymbol{z}z}^{\boldsymbol{t}t}}$. Otherwise, $\varphi_C^I(\boldsymbol{t}, \boldsymbol{S}) = \mathcal{SN}$.*

The fact that φ is monotone, hence has a least fixpoint, follows from the positivity conditions. One can easily check that φ_C^I is stable by reduction: if $\boldsymbol{t} \rightarrow \boldsymbol{t}'$ then $\varphi_C^I(\boldsymbol{t}, \boldsymbol{S}) = \varphi_C^I(\boldsymbol{t}', \boldsymbol{S})$. We now prove that $\varphi_C^I(\boldsymbol{t}, \boldsymbol{S})$ is a candidate.

Lemma 3. *$\varphi_C^I(\boldsymbol{t}, \boldsymbol{S})$ is a candidate.*

Proof. (R1) Let $t \in R$. We must prove that $t \in \mathcal{SN}$. Since $\mathcal{R}ec(C) \neq \emptyset$, there is at least one recursor f. Take $y_i\theta = y_i$ and $y_i\xi = \top_{U_i}$. We clearly have $\xi_{\boldsymbol{z}}^{\boldsymbol{S}}, \theta_{\boldsymbol{z}z}^{\boldsymbol{t}t} \models \boldsymbol{y} : \boldsymbol{U}$. Therefore, $f\boldsymbol{t}^*t\boldsymbol{y} \in S = [\![V]\!]^I_{\xi_{\boldsymbol{z}}^{\boldsymbol{S}}, \theta_{\boldsymbol{z}z}^{\boldsymbol{t}t}}$. Now, since S satisfies (R1), $f\boldsymbol{t}^*t\boldsymbol{y} \in \mathcal{SN}$ and $t \in \mathcal{SN}$.

(R2) Let $t \in R$ and $t' \in \rightarrow(t)$. We must prove that $t' \in R$, hence that $f\boldsymbol{t}^*t'\boldsymbol{y}\theta \in S = [\![V]\!]^I_{\xi_{\boldsymbol{z}}^{\boldsymbol{S}}, \theta_{\boldsymbol{z}z}^{\boldsymbol{t}t}}$. This follows from the fact that $f\boldsymbol{t}^*t\boldsymbol{y}\theta \in S$ (since $t \in R$) and S satisfies (R2).

(R3) Let t be a neutral term such that $\rightarrow(t) \subseteq R$. We must prove that $t \in R$, hence that $u = f\boldsymbol{t}^*t\boldsymbol{y}\theta \in S = [\![V]\!]^I_{\xi_{\boldsymbol{z}}^{\boldsymbol{S}}, \theta_{\boldsymbol{z}z}^{\boldsymbol{t}t}}$. Since u is neutral and S satisfies (R3), it suffices to prove that $\rightarrow(u) \subseteq S$. Since $\boldsymbol{y}\theta \in \mathcal{SN}$ by (R1), we proceed by induction on $\boldsymbol{y}\theta$ with $\rightarrow$ as well-founded ordering. The only difficult case could be when u is head-reducible, but this is not possible since t is neutral, hence not constructor-headed. $\Box$

5 Admissible recursors

Since we changed the interpretation of constant predicate symbols, we must check several things in order to preserve the strong normalization result of [5].

- We must make sure that the interpretation of primitive types is still $\mathcal{SN}$ since this is used for proving the computability of first-order symbols and the interpretation of some defined predicate symbols (see Lemma 5).

[5] In Section 8, we give weaker conditions for dealing with inductive-recursive types.

- We must also prove that every symbol is computable.
  - For extended recursors, this follows from the definition of the interpretation for constant predicate symbols, and thus, does not require safety.
  - For first-order symbols, nothing is changed.
  - For higher-order symbols distinct from recursors, we must make sure that the accessible arguments of a computable constructor-headed term are computable.
  - For constructors, this does not follow from the interpretation for constant predicate symbols anymore. We therefore have to prove it.

We now define general conditions for these requirements to be satisfied.

Definition 4 (Admissible recursors). *Assume that every constructor is equipped with a set* $\mathrm{Acc}(c) \subseteq \{1, \ldots, \alpha_c\}$ *of* accessible arguments*. Let* $C : (\boldsymbol{z} : \boldsymbol{V})\star$ *be a constant predicate symbol.* $\mathcal{R}ec(C)$ *is* complete w.r.t. accessibility *if, for all* $c : (\boldsymbol{x} : \boldsymbol{T})C\boldsymbol{v}$*,* $j \in \mathrm{Acc}(c)$*,* $\boldsymbol{x}\eta$ *and* $\boldsymbol{x}\sigma$*, if* $\eta \models \Gamma_c$*,* $\boldsymbol{v}\sigma \in \mathcal{SN}$ *and* $c\boldsymbol{x}\sigma \in [\![C\boldsymbol{v}]\!]_{\eta,\sigma}$ *then* $x_j\sigma \in [\![T_j]\!]_{\eta,\sigma}$*.*

A recursor $f : (\boldsymbol{z} : \boldsymbol{V})(z : C\boldsymbol{z})(\boldsymbol{y} : \boldsymbol{U})V$ *is* head-computable w.r.t *a constructor* $c : (\boldsymbol{x} : \boldsymbol{T})C\boldsymbol{v}$ *if, for all* $\boldsymbol{x}\eta$*,* $\boldsymbol{x}\sigma$*,* $\boldsymbol{y}\xi$*,* $\boldsymbol{y}\theta$*,* $\boldsymbol{S} = [\![\boldsymbol{v}]\!]_{\eta,\sigma}$*, if* $\eta, \sigma \models \Gamma_c$ *and* $\xi_{\boldsymbol{z}}^{\boldsymbol{S}}, \theta_{\boldsymbol{z}}^{\boldsymbol{v}\sigma}{}_{z}^{c\boldsymbol{x}\sigma} \models \boldsymbol{y} : \boldsymbol{U}$*, then every head-reduct of* $f\boldsymbol{v}\sigma(c\boldsymbol{x}\sigma)\boldsymbol{y}\theta$ *belongs to* $[\![V]\!]_{\xi_{\boldsymbol{z}}^{\boldsymbol{S}}, \theta_{\boldsymbol{z}}^{\boldsymbol{v}\sigma}{}_{z}^{c\boldsymbol{x}\sigma}}$*. A recursor is* head-computable *if it is head-computable w.r.t. every constructor.* $\mathcal{R}ec(C)$ *is* head-computable *if all its recursors are head-computable.*

$\mathcal{R}ec(C)$ *is* admissible *if it is head-computable and complete w.r.t. accessibility.*

We first prove that the interpretation of primitive types is $\mathcal{SN}$.

Lemma 5 (Primitive types). *Types equivalent to* C *are* primitive *if, for all* $D \simeq C$*,* $D : \star$ *and, for all* $d : (\boldsymbol{x} : \boldsymbol{T})D$*,* $\mathrm{Acc}(d) = \{1, \ldots, \alpha_d\}$ *and every* T_j *is a primitive type* $E \leq C$*. Let* $C : \star$ *be a primitive symbol. If recursors are head-computable then* $I_C = \mathcal{SN}$*.*

Proof. By definition, $I_C \subseteq \mathcal{SN}$. We prove that, if $t \in \mathcal{SN}$ then $t \in I_C$, by induction on t with $\rightarrow \cup \rhd$ as well-founded ordering. Let $f : (z : C)(\boldsymbol{y} : \boldsymbol{U})V$ be a recursor, $\boldsymbol{y}\xi$ and $\boldsymbol{y}\theta$ such that $\xi, \theta_z^t \models \boldsymbol{y} : \boldsymbol{U}$. We must prove that $v = ft\boldsymbol{y}\theta \in S = [\![V]\!]_{\xi,\theta_z^t}$. Since v is neutral, it suffices to prove that $\rightarrow(v) \subseteq S$. We proceed by induction on $t\boldsymbol{y}\theta$ with $\rightarrow$ as well-founded ordering ($\boldsymbol{y}\theta \in \mathcal{SN}$ by R1). If the reduction takes place in $t\boldsymbol{y}\theta$, we can conclude by induction hypothesis. Assume now that v' is a head-reduct of v. By assumption on recursors (Definition 1), t is of the form $c\boldsymbol{u}$ with $c : (\boldsymbol{x} : \boldsymbol{T})C$. Since C is primitive, every u_j is accessible and every T_j is a primitive type $D \leq C$. By induction hypothesis, $u_j \in I_D$. Therefore, $\emptyset, \{\boldsymbol{x} \mapsto \boldsymbol{u}\} \models \Gamma_c$ and, since $\xi, \theta_z^t \models \boldsymbol{y} : \boldsymbol{U}$ and recursors are head-computable, $v' \in S$. $\square$

Theorem 6 (Strong normalization). *Assume that every constant predicate symbol* C *is equipped with an admissible set* $\mathcal{R}ec(C)$ *of extended recursors distinct from constructors. If* $\rightarrow$ *is confluent and strong recursors and symbols that are not recursors satisfy the conditions given in [5] then* $\beta \cup \mathcal{R}$ *is strongly normalizing.*

Proof. Let $\vdash_f$ (resp. $\vdash_f^<$) be the typing relation of the CAC whose symbols are (resp. strictly) smaller than f. By induction on f, we prove that, if $\Gamma \vdash_f t : T$ and $\xi, \theta \models \Gamma$ then $t\theta \in [\![T]\!]_{\xi,\theta}$. By (symb), if $g \leq f$ and $\vdash_f g : \tau_g$ then $\vdash_f^< \tau_g : s_g$. Therefore, the induction hypothesis can be applied to the subterms of τ_g.

We first prove that recursors are computable. Let $f : (\boldsymbol{z} : \boldsymbol{V})(z : C\boldsymbol{z})(\boldsymbol{y} : \boldsymbol{U})V$ be a recursor and assume that $\xi, \theta \models \Gamma_f$. We must prove that $v = f\boldsymbol{z}\theta z\theta \boldsymbol{y}\theta \in S = [\![V]\!]_{\xi,\theta}$. Since v is neutral, it suffices to prove that $\rightarrow(v) \subseteq S$. We proceed by induction on $\boldsymbol{z}\theta z\theta \boldsymbol{y}\theta$ with $\rightarrow$ as well-founded ordering ($\boldsymbol{z}\theta z\theta \boldsymbol{y}\theta \in \mathcal{SN}$ by R1). If the reduction takes place in $\boldsymbol{z}\theta z\theta \boldsymbol{y}\theta$, we conclude by induction hypothesis. Assume now that we have a head-reduct v'. By assumption on recursors (Definition 1), $z\theta$ is of the form $c\boldsymbol{u}$ with $c : (\boldsymbol{x} : \boldsymbol{T})C\boldsymbol{v}$, and v' is a head-reduct of $v_0 = f\boldsymbol{z}\theta^* z\theta \boldsymbol{y}\theta$ where $\boldsymbol{z}\theta^*$ are the normal forms of $\boldsymbol{z}\theta$. Since $\xi, \theta \models \Gamma_f$, we have $z\theta = c\boldsymbol{u} \in [\![C\boldsymbol{z}]\!]_{\xi,\theta} = I_C(\boldsymbol{z}\theta, \boldsymbol{z}\xi)$. Therefore, $v_0 \in S$ and, by (R2), $v' \in S$.

We now prove that constructors are computable. Let $c : (\boldsymbol{x} : \boldsymbol{T})C\boldsymbol{v}$ be a constructor of $C : (\boldsymbol{z} : \boldsymbol{V})\star$, $\boldsymbol{x}\eta$ and $\boldsymbol{x}\sigma$ such that $\eta, \sigma \models \Gamma_c$. We must prove that $c\boldsymbol{x}\sigma \in [\![C\boldsymbol{v}]\!]_{\eta,\sigma} = I_C(\boldsymbol{v}\sigma, \boldsymbol{S})$ where $\boldsymbol{S} = [\![\boldsymbol{v}]\!]_{\eta,\sigma}$. By induction hypothesis, we have $\boldsymbol{v}\sigma \in \mathcal{SN}$. So, let $f : (\boldsymbol{z} : \boldsymbol{V})(z : C\boldsymbol{z})(\boldsymbol{y} : \boldsymbol{U})V$ be a recursor of C, $\boldsymbol{y}\xi$ and $\boldsymbol{y}\theta$ such that $\xi_{\boldsymbol{z}}^{\boldsymbol{S}}, \theta_{\boldsymbol{z}\ z}^{\boldsymbol{v}\sigma\, c\boldsymbol{x}\sigma} \models \boldsymbol{y} : \boldsymbol{U}$. We must prove that $v = f\boldsymbol{v}\sigma^*(c\boldsymbol{x}\sigma)\boldsymbol{y}\theta \in S = [\![V]\!]_{\xi_{\boldsymbol{z}}^{\boldsymbol{S}}, \theta_{\boldsymbol{z}\ z}^{\boldsymbol{v}\sigma\, c\boldsymbol{x}\sigma}}$. Since v is neutral, it suffices to prove that $\rightarrow(v) \subseteq S$. Since $\boldsymbol{y}\theta \in \mathcal{SN}$, we can proceed by induction on $\boldsymbol{y}\theta$ with $\rightarrow$ as well-founded ordering.

In the case of a reduction in $\boldsymbol{y}\theta$, we conclude by induction hypothesis. In the case of a head-reduction, we conclude by head-computability of f. And, in the case of a reduction in $c\boldsymbol{x}\sigma$, we conclude by the computability lemmas for function symbols in [5]: if the strong normalization conditions are satisfied and accessibility is correct w.r.t. computability, then every reduct of $c\boldsymbol{x}\sigma$ belongs to $[\![C\boldsymbol{v}]\!]_{\eta,\sigma}$. The fact that accessibility is correct w.r.t. computability follows from the completeness of the set of recursors w.r.t. accessibility. □

6 The Calculus of Inductive Constructions

As an example, we prove the admissibility of a large class of weak recursors for strictly positive types, from which Coq's recursors [22] can be easily derived. This can be extended to strong recursors and to some non-strictly positive types (see Section 7).

Definition 7. *Let $C : (\boldsymbol{z} : \boldsymbol{V})\star$ and $\boldsymbol{c}$ be* strictly positive *constructors of C, that is, if c_i is of type $(\boldsymbol{x} : \boldsymbol{T})C\boldsymbol{v}$ then either no type equivalent to C occurs in T_j or T_j is of the form $(\boldsymbol{\alpha} : \boldsymbol{W})C\boldsymbol{w}$ with no type equivalent to C occurring in $\boldsymbol{W}$. The* parameters *of C is the biggest sequence $\boldsymbol{q}$ such that $C : (\boldsymbol{q} : \boldsymbol{Q})(\boldsymbol{z} : \boldsymbol{V})\star$ and each c_i is of type $(\boldsymbol{q} : \boldsymbol{Q})(\boldsymbol{x} : \boldsymbol{T})C\boldsymbol{q}\boldsymbol{v}$ with $T_j = (\boldsymbol{\alpha} : \boldsymbol{W})C\boldsymbol{q}\boldsymbol{w}$ if C occurs in T_j.*

The canonical *weak recursor*[6] *of C w.r.t $\boldsymbol{c}$ is $rec^*_{\boldsymbol{c}} : (\boldsymbol{q} : \boldsymbol{Q})(\boldsymbol{z} : \boldsymbol{V})(z : C\boldsymbol{q}\boldsymbol{z})(P : (\boldsymbol{z} : \boldsymbol{V})C\boldsymbol{q}\boldsymbol{z} \Rightarrow \star)(\boldsymbol{y} : \boldsymbol{U})P\boldsymbol{z}z$ with $U_i = (\boldsymbol{x} : \boldsymbol{T})(\boldsymbol{x}' : \boldsymbol{T}')P\boldsymbol{v}(c_i\boldsymbol{q}\boldsymbol{x})$, $T'_j = (\boldsymbol{\alpha} : \boldsymbol{W})P\boldsymbol{w}(x_j\boldsymbol{\alpha})$ if $T_j = (\boldsymbol{\alpha} : \boldsymbol{W})C\boldsymbol{q}\boldsymbol{w}$, and $T'_j = T_j$ otherwise, defined*

[6] Strong recursors cannot be defined by taking $P : (\boldsymbol{z} : \boldsymbol{V})C\boldsymbol{q}\boldsymbol{z} \Rightarrow \square$ instead since $(\boldsymbol{z} : \boldsymbol{V})C\boldsymbol{q}\boldsymbol{z} \Rightarrow \square$ is not typable in CC. They must be defined for each P.

by the rules $rec^*_{\boldsymbol{c}}\boldsymbol{q}\boldsymbol{z}(c_i\boldsymbol{q}'\boldsymbol{x})P\boldsymbol{y} \rightarrow y_i\boldsymbol{x}\boldsymbol{t}'$ *where* $t'_j = [\boldsymbol{\alpha} : \boldsymbol{W}](rec^*_{\boldsymbol{c}}\boldsymbol{q}\boldsymbol{w}(x_j\boldsymbol{\alpha})P\boldsymbol{y})$ *if* $T_j = (\boldsymbol{\alpha} : \boldsymbol{W})C\boldsymbol{q}\boldsymbol{w}$*, and* $t'_j = x_j$ *otherwise.*[7]

Lemma 8. *The set of canonical recursors is complete w.r.t. accessibility.*[8]

Proof. Let $c = c_i : (\boldsymbol{q} : \boldsymbol{Q})(\boldsymbol{x} : \boldsymbol{T})C\boldsymbol{q}\boldsymbol{v}$ be a constructor of $C : (\boldsymbol{q} : \boldsymbol{Q})(\boldsymbol{z} : \boldsymbol{V})\star$, $\boldsymbol{q}\eta$, $\boldsymbol{x}\eta$, $\boldsymbol{q}\sigma$ and $\boldsymbol{x}\sigma$ such that $\boldsymbol{q}\sigma\boldsymbol{v}\sigma \in \mathcal{SN}$ and $c\boldsymbol{q}\sigma\boldsymbol{x}\sigma \in [\![C\boldsymbol{q}\boldsymbol{v}]\!]_{\eta,\sigma} = I_C(\boldsymbol{q}\sigma\boldsymbol{v}\sigma, \boldsymbol{q}\xi[\![\boldsymbol{v}]\!]_{\eta,\sigma})$. Let $\boldsymbol{a} = \boldsymbol{q}\boldsymbol{x}$ and $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{T}$. We must prove that, for all j, $a_j\sigma \in [\![A_j]\!]_{\eta,\sigma}$. For the sake of simplicity, we assume that weak and strong recursors have the same syntax. Since $\boldsymbol{q}\sigma\boldsymbol{v}\sigma$ have normal forms, it suffices to find u_j such that $rec_c\boldsymbol{q}\boldsymbol{v}(c\boldsymbol{q}\boldsymbol{x})P_ju_j \rightarrow u_j\boldsymbol{x}\boldsymbol{t}' \rightarrow^*_\beta a_j$. Take $u_j = [\boldsymbol{x} : \boldsymbol{T}][\boldsymbol{x}' : \boldsymbol{T}']a_j$. ☐

Lemma 9. *Canonical recursors are head-computable.*

Proof. Let $f : (\boldsymbol{q} : \boldsymbol{Q})(\boldsymbol{z} : \boldsymbol{V})(z : C\boldsymbol{q}\boldsymbol{z})(P : (\boldsymbol{z} : \boldsymbol{V})C\boldsymbol{q}\boldsymbol{z} \Rightarrow \star)(\boldsymbol{y} : \boldsymbol{U})P\boldsymbol{z}z$ be the canonical weak recursor w.r.t. $\boldsymbol{c}$, $T = (\boldsymbol{z} : \boldsymbol{V})C\boldsymbol{q}\boldsymbol{z} \Rightarrow \star$, $c = c_i : (\boldsymbol{q} : \boldsymbol{Q})(\boldsymbol{x} : \boldsymbol{T})C\boldsymbol{q}\boldsymbol{v}$, $\boldsymbol{q}\eta$, $\boldsymbol{q}\sigma$, $\boldsymbol{x}\eta$, $\boldsymbol{x}\sigma$, $P\xi$, $P\theta$, $\boldsymbol{y}\xi$, $\boldsymbol{y}\theta$, $\boldsymbol{R} = [\![\boldsymbol{v}]\!]_{\eta,\sigma}$, $\xi' = \xi^{\boldsymbol{R}}_{\boldsymbol{z}}$ and $\theta' = \theta^{\boldsymbol{v}\sigma}_{\boldsymbol{z}}{}^{c\boldsymbol{x}\sigma}_{z}$, and assume that $\eta, \sigma \models \Gamma_c$ and $\eta\xi', \sigma\theta' \models P : T, \boldsymbol{y} : \boldsymbol{U}$. We must prove that $y_i\theta\boldsymbol{x}\sigma\boldsymbol{t}'\sigma\theta \in [\![P\boldsymbol{z}z]\!]_{\xi',\theta'}$.

We have $y_i\theta \in [\![U_i]\!]_{\xi',\theta'}$, $U_i = (\boldsymbol{x} : \boldsymbol{T})(\boldsymbol{x}' : \boldsymbol{T}')P\boldsymbol{v}(c\boldsymbol{q}\boldsymbol{x})$ and $x_j\sigma \in [\![T_j]\!]_{\eta,\sigma} = [\![T_j]\!]_{\eta\xi',\sigma\theta'}$. We prove that $t'_j\sigma\theta \in [\![T'_j]\!]_{\eta\xi',\sigma\theta'}$. If $T'_j = T_j$ then $t'_j\sigma\theta = x_j\sigma$ and we are done. Otherwise, $T_j = (\boldsymbol{\alpha} : \boldsymbol{W})C\boldsymbol{q}\boldsymbol{w}$, $T'_j = (\boldsymbol{\alpha} : \boldsymbol{W})P\boldsymbol{w}(x_j\boldsymbol{\alpha})$ and $t'_j = [\boldsymbol{\alpha} : \boldsymbol{W}]f\boldsymbol{q}\boldsymbol{w}(x_j\boldsymbol{\alpha})P\boldsymbol{y}$. Let $\boldsymbol{\alpha}\zeta$ and $\boldsymbol{\alpha}\gamma$ such that $\eta\xi'\zeta, \sigma\theta'\gamma \models \boldsymbol{\alpha} : \boldsymbol{W}$. Let $t = x_j\sigma\boldsymbol{\alpha}\gamma$. We must prove that $v = f\boldsymbol{q}\sigma\boldsymbol{w}\sigma\gamma tP\theta\boldsymbol{y}\theta \in S = [\![P\boldsymbol{w}(x_j\boldsymbol{\alpha})]\!]_{\eta\xi'\zeta,\sigma\theta'\gamma}$. Since v is neutral, it suffices to prove that $\rightarrow(v) \subseteq S$.

We proceed by induction on $\boldsymbol{q}\sigma\boldsymbol{w}\sigma\gamma tP\theta\boldsymbol{y}\theta \in \mathcal{SN}$ with $\rightarrow$ as well-founded ordering (we can assume that $\boldsymbol{w}\sigma\gamma \in \mathcal{SN}$ since $\vdash^<_f \tau_f : s_f$). In the case of a reduction in $\boldsymbol{q}\sigma\boldsymbol{w}\sigma\gamma tP\theta\boldsymbol{y}\theta$, we conclude by induction hypothesis. Assume now that we have a head-reduct v'. By definition of recursors, v' is also a head-reduct of $v_0 = f\boldsymbol{q}\sigma^*\boldsymbol{w}\sigma\gamma^*tP\theta\boldsymbol{y}\theta$ where $\boldsymbol{q}\sigma^*\boldsymbol{w}\sigma\gamma^*$ are the normal forms of $\boldsymbol{q}\sigma\boldsymbol{w}\sigma\gamma$. If $v_0 \in S$ then, by (R2), $v' \in S$. So, let us prove that $v_0 \in S$.

By candidate substitution, $S = [\![P\boldsymbol{z}z]\!]_{\xi^{\boldsymbol{S}}_{\boldsymbol{z}},\theta^{\boldsymbol{w}\sigma\gamma}_{\boldsymbol{z}}{}^{t}_{z}}$ with $\boldsymbol{S} = [\![\boldsymbol{w}]\!]_{\eta\xi'\zeta,\sigma\theta'\gamma} = [\![\boldsymbol{w}]\!]_{\eta\xi\zeta,\sigma\theta\gamma}$ for $\mathrm{FV}(\boldsymbol{w}) \subseteq \{\boldsymbol{q}, P, \boldsymbol{x}, \boldsymbol{\alpha}\}$. Since $x_j\sigma \in [\![T_j]\!]_{\eta\xi',\sigma\theta'}$ and $\eta\xi'\zeta, \sigma\theta'\gamma \models \boldsymbol{\alpha} : \boldsymbol{W}$, $t \in [\![C\boldsymbol{q}\boldsymbol{w}]\!]_{\eta\xi'\zeta,\sigma\theta'\gamma} = I_C(\boldsymbol{q}\sigma\boldsymbol{w}\sigma\gamma, \boldsymbol{q}\xi\boldsymbol{S})$. Since $\eta\xi', \sigma\theta' \models P : T, \boldsymbol{y} : \boldsymbol{U}$ and $\mathrm{FV}(T\boldsymbol{U}) \subseteq \{\boldsymbol{q}, P\}$, we have $\eta\xi, \sigma\theta \models P : T, \boldsymbol{y} : \boldsymbol{U}$ and $\eta\xi^{\boldsymbol{S}}_{\boldsymbol{z}}, \sigma\theta^{\boldsymbol{w}\sigma\gamma}_{\boldsymbol{z}}{}^{t}_{z} \models P : T, \boldsymbol{y} : \boldsymbol{U}$. Therefore, $v_0 \in S$. ☐

It follows that CAC essentially subsumes CIC as defined in [23]. Theorem 6 cannot be applied to CIC directly since CIC and CAC do not have the same syntax and the same typing rules. So, in [5], we defined a sub-system of CIC, called CIC$^-$, whose terms can be translated into a CAC. Without requiring inductive types to be *safe* and to satisfy (I6), we think that CIC$^-$ is essentially as powerful as CIC.

Theorem 10. *The system CIC$^-$ defined in [5] (Chapter 7) is strongly normalizing even though inductive types are unsafe and do not satisfy (I6).*

[7] We could erase the useless arguments $t'_j = x_j$ when $T'_j = T_j$.

[8] In [23] (Lemma 4.35), Werner proves a similar result.

7 Non-strictly positive types

We are going to see that the use of elimination-based interpretations allows us to have functions defined by recursion on non-strictly positive types too, while CIC has always been restricted to strictly positive types. An interesting example is given by Abel's formalization of first-order terms with continuations as an inductive type $trm : \star$ with the constructors [1]:

$$\begin{array}{l} var : nat \Rightarrow trm \\ fun : nat \Rightarrow (list\ trm) \Rightarrow trm \\ mu : \neg\neg trm \Rightarrow trm \end{array}$$

where $list : \star \Rightarrow \star$ is the type of polymorphic lists, $\neg X$ is an abbreviation for $X \Rightarrow \bot$ (in the next section, we prove that $\neg$ can be defined as a function), and $\bot : \star$ is the empty type. Its recursor $rec : (A : \star)(y_1 : nat \Rightarrow A)\ (y_2 : nat \Rightarrow list\ trm \Rightarrow listA \Rightarrow A)(y_3 : \neg\neg trm \Rightarrow \neg\neg A \Rightarrow A)(z : trm)A$ can be defined by:

$$\begin{array}{rcl} rec\ A\ y_1\ y_2\ y_3\ (var\ n) & \to & y_1\ n \\ rec\ A\ y_1\ y_2\ y_3\ (fun\ n\ l) & \to & y_2\ n\ l\ (map\ trm\ A\ (rec\ A\ y_1\ y_2\ y_3)\ l) \\ rec\ A\ y_1\ y_2\ y_3\ (mu\ f) & \to & y_3\ f\ [x : \neg A](f\ [y : trm](x\ (rec\ A\ y_1\ y_2\ y_3\ y))) \end{array}$$

where $map : (A : \star)(B : \star)(A \Rightarrow B) \Rightarrow list\ A \Rightarrow list\ B$ is defined by:

$$\begin{array}{rcl} map\ A\ B\ f\ (nil\ A') & \to & (nil\ B) \\ map\ A\ B\ f\ (cons\ A'\ x\ l) & \to & cons\ B\ (f\ x)\ (map\ A\ B\ f\ l) \\ map\ A\ B\ f\ (app\ A'\ l\ l') & \to & app\ B\ (map\ A\ B\ f\ l)\ (map\ A\ B\ f\ l') \end{array}$$

We now check that rec is an admissible recursor. Completeness w.r.t. accessibility is easy. For the head-computability, we only detail the case of mu. Let $f\sigma$, $t = mu\ f\sigma$, $A\xi$, $A\theta$ and $\boldsymbol{y}\theta$ such that $\emptyset, \sigma \models \Gamma_{mu}$ and $\xi, \sigma\theta^t_z \models \Gamma = A : \star$, $\boldsymbol{y} : \boldsymbol{U}$ where U_i is the type of y_i. Let $b = recA\theta\boldsymbol{y}\theta$, $c = [y : trm](x(by))$ and $a = [x : \neg A\theta](f\sigma c)$. We must prove that $y_3\theta f\sigma a \in [\![A]\!]_{\xi,\sigma\theta^t_z} = A\xi$.

Since $\xi, \sigma\theta^t_z \models \Gamma$, $y_3\theta \in [\![\neg\neg trm \Rightarrow \neg\neg A \Rightarrow A]\!]_{\xi,\theta}$. Since $\emptyset, \sigma \models \Gamma_{mu}$, $f\sigma \in [\![\neg\neg trm]\!]$. Thus, we are left to prove that $a \in [\![\neg\neg A]\!]_{\xi,\theta}$, that is, $f\sigma c\gamma \in I_\bot$ for all $x\gamma \in [\![\neg A]\!]_{\xi,\theta}$. Since $f\sigma \in [\![\neg\neg trm]\!]$, it suffices to prove that $c\gamma \in [\![\neg trm]\!]$, that is, $x\gamma(by\gamma) \in I_\bot$ for all $y\gamma \in I_{trm}$. This follows from the facts that $x\gamma \in [\![\neg A]\!]_{\xi,\theta}$ and $by\gamma \in A\xi$ since $y\gamma \in I_{trm}$.

8 Inductive-recursive types

In this section, we define new positivity conditions for dealing with *inductive-recursive type definitions* [12]. An inductive-recursive type C has constructors whose arguments have a type Ft with F defined by recursion on $t : C$, that is, a predicate F and its domain C are defined at the same time.

A simple example is the type $dlist : (A : \star)(\# : A \Rightarrow A \Rightarrow \star)\star$ of lists made of distinct elements thanks to the predicate $fresh : (A : \star)(\# : A \Rightarrow A \Rightarrow \star)A \Rightarrow (dlist\,A\,\#) \Rightarrow \star$ parametrized by a function $\#$ to test whether two elements are distinct. The constructors of $dlist$ are:

$$
\begin{array}{l}
\quad nil : (A:\star)(\#:A \Rightarrow A \Rightarrow \star)(dlist\, A\, \#)\\
cons : (A:\star)(\#:A \Rightarrow A \Rightarrow \star)(x:A)(l: dlist\, A\, \#)(fresh\ A\ \#\ x\ l) \Rightarrow (dlist\, A\, \#)
\end{array}
$$

and the rules defining $fresh$ are:

$$
\begin{array}{rcl}
fresh\ A\ \#\ x\ (nil\ A') & \to & \top\\
fresh\ A\ \#\ x\ (cons\ A'\ y\ l\ h) & \to & x\#y \wedge fresh\ A\ \#\ x\ l
\end{array}
$$

where $\top$ is the proposition always true and $\wedge$ the connector "and". Other examples are given by Martin-Löf's definition of the first universe *à la* Tarski [12] or by Pollack's formalization of record types with manifest fields [21].

Definition 11 (Positive/negative positions). *Assume that every predicate symbol* $f : (\boldsymbol{x} : \boldsymbol{t})U$ *is equipped with a set* $\mathrm{Mon}^+(f) \subseteq \{i \le \alpha_f \mid x_i \in \mathcal{X}^\Box\}$ *of* monotone arguments *and a set* $\mathrm{Mon}^-(f) \subseteq \{i \le \alpha_f \mid x_i \in \mathcal{X}^\Box\}$ *of* anti-monotone arguments. *The sets of* positive positions $\mathrm{Pos}^+(t)$ *and* negative positions $\mathrm{Pos}^-(t)$ *in a term* t *are inductively defined as follows:*

- $\mathrm{Pos}^\delta(s) = \mathrm{Pos}^\delta(x) = \{\varepsilon \mid \delta = +\}$,
- $\mathrm{Pos}^\delta((x:U)V) = 1.\mathrm{Pos}^{-\delta}(U) \cup 2.\mathrm{Pos}^\delta(V)$,
- $\mathrm{Pos}^\delta([x:U]v) = 2.\mathrm{Pos}^\delta(v)$,
- $\mathrm{Pos}^\delta(tu) = 1.\mathrm{Pos}^\delta(t)$ *if* $t \neq f\boldsymbol{t}$,
- $\mathrm{Pos}^\delta(f\boldsymbol{t}) = \{1^{|\boldsymbol{t}|} \mid \delta = +\} \cup \bigcup \{1^{|\boldsymbol{t}|-i}2.\mathrm{Pos}^{\epsilon\delta}(t_i) \mid \epsilon \in \{-,+\},\, i \in \mathrm{Mon}^\epsilon(f)\}$,

where $\delta \in \{-,+\}$, $-+ = -$ *and* $-- = +$ *(usual rule of signs).*

Theorem 12 (Strong normalization). *Definition 1 is modified as follows. A pre-recursor* $f : (\boldsymbol{z} : \boldsymbol{V})(z : C\boldsymbol{z})W$ *is a* recursor *if:*

- *no* $F > C$ *occurs in* W,
- *every* $F \simeq C$ *occurs only positively in* W,
- *if* $i \in \mathrm{Mon}^\delta(C)$ *then* $\mathrm{Pos}(z_i, W) \subseteq \mathrm{Pos}^\delta(W)$.

Assume furthermore that, for every rule $F\boldsymbol{l} \to r$*:*

- *no* $G > F$ *occurs in* r,
- *for all* $i \in \mathrm{Mon}^\epsilon(F)$, $l_i \in \mathcal{X}^\Box$ *and* $\mathrm{Pos}(l_i, r) \subseteq \mathrm{Pos}^\epsilon(r)$.

Then, Theorem 6 is still valid.

Proof. For Theorem 6 to be still valid, we must make sure that φ (see Definition 2) is still monotone, hence has a least fixpoint. To this end, we need to prove that $[\![t]\!]^I_{\xi,\theta}$ is monotone (resp. anti-monotone) w.r.t. $x\xi$ if x occurs only positively (resp. negatively) in t, and that $[\![t]\!]^I_{\xi,\theta}$ is monotone (resp. anti-monotone) w.r.t. I_C if C occurs only positively (resp. negatively) in t. These results are easily extended to the new positivity conditions by reasoning by induction on the well-founded ordering used for defining the defined predicate symbols.

Let us see what happens in the case where $t = F\boldsymbol{t}$ with F a defined predicate symbol. Let $\le^+ = \le$ and $\le^- = \ge$. We want to prove that, if $\xi_1 \le_x \xi_2$ (*i.e.* $x\xi_1 \le x\xi_2$ and, for all $y \neq x$, $y\xi_1 = y\xi_2$) and $\mathrm{Pos}(x,t) \subseteq \mathrm{Pos}^\delta(t)$, then $[\![t]\!]^I_{\xi_1,\theta} \le^\delta [\![t]\!]^I_{\xi_2,\theta}$. By definition of I_F, if the normal forms of $\boldsymbol{t}\theta$ matches the left hand-side of

$\boldsymbol{Fl} \to r$, then $[\![F\boldsymbol{t}]\!]^I_{\xi_i,\theta} = [\![r]\!]^I_{\xi'_i,\sigma}$ where σ is the matching substitution and, for all $y \in \mathrm{FV}(r)$, $y\xi'_i = [\![t_{\kappa_y}]\!]^I_{\xi_i,\theta}$ where κ_y is such that $l_{\kappa_y} = y$ (see [5] for details). Now, since $\mathrm{Pos}(x, F\boldsymbol{t}) \subseteq \mathrm{Pos}^\delta(F\boldsymbol{t})$, $\mathrm{Pos}(x, t_{\kappa_y}) \subseteq \mathrm{Pos}^{\epsilon\delta}(t_{\kappa_y})$ for some ϵ. Hence, by induction hypothesis, $\xi'_1 \leq_y^{\epsilon\delta} \xi'_2$. Now, since $\mathrm{Pos}(y, r) \subseteq \mathrm{Pos}^\epsilon(r)$, by induction hypothesis again, $[\![r]\!]_{\xi'_1,\sigma} \leq^{\epsilon^2\delta} = \leq^\delta [\![r]\!]_{\xi'_2,\sigma}$. $\square$

For instance, in the positive type trm of Section 7, instead of considering $\neg\neg A$ as an abbreviation, one can consider $\neg$ as a predicate symbol defined by the rule $\neg A \to A \Rightarrow \bot$ with $\mathrm{Mon}^-(\neg) = \{1\}$. Then, one easily checks that A occurs negatively in $A \Rightarrow \bot$, and hence that trm occurs positively in $\neg\neg trm$ since $\mathrm{Pos}^+(\neg\neg trm) = \{1\} \cup 2.\mathrm{Pos}^-(\neg trm) = \{1\} \cup 2.2.\mathrm{Pos}^+(trm) = \{1, 2.2\}$.

9 Conclusion

By using an elimination-based interpretation for inductive types, we proved that the Calculus of Algebraic Constructions completely subsumes the Calculus of Inductive Constructions. We define general conditions on extended recursors for preserving strong normalization and show that these conditions are satisfied by a large class of recursors for strictly positive types and by non-strictly positive types too. Finally, we give general positivity conditions for dealing with inductive-recursive types.

Acknowledgments. I would like to thank C. Paulin, R. Matthes, J.-P. Jouannaud, D. Walukiewicz, G. Dowek and the anonymous referees for their useful comments on previous versions of this paper. Part of this work was performed during my stay at Cambridge (UK) thanks to a grant from the INRIA.

References

1. A. Abel. Termination checking with types. Technical Report 0201, Ludwig Maximilians Universität, München, Germany, 2002.
2. F. Barbanera, M. Fernández, and H. Geuvers. Modularity of strong normalization and confluence in the algebraic-λ-cube. In *Proceedings of the 9th IEEE Symposium on Logic in Computer Science*, 1994.
3. H. Barendregt. Lambda calculi with types. In S. Abramski, D. Gabbay, and T. Maibaum, editors, *Handbook of logic in computer science*, volume 2. Oxford University Press, 1992.
4. F. Blanqui. Definitions by rewriting in the Calculus of Constructions (extended abstract). In *Proceedings of the 16th IEEE Symposium on Logic in Computer Science*, 2001.
5. F. Blanqui. *Théorie des Types et Récriture*. PhD thesis, Université Paris XI, Orsay, France, 2001. Available in english as "Type Theory and Rewriting".
6. F. Blanqui. Definitions by rewriting in the Calculus of Constructions, 2003. Journal submission, 68 pages.
7. F. Blanqui. A short and flexible strong normalization proof for the Calculus of Algebraic Constructions with curried rewriting, 2003. Draft.

8. T. Coquand. Pattern matching with dependent types. In *Proceedings of the International Workshop on Types for Proofs and Programs, 1992*. `http://www.lfcs.informatics.ed.ac.uk/research/types-bra/proc/`.
9. T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76(2–3):95–120, 1988.
10. T. Coquand and C. Paulin-Mohring. Inductively defined types. In *Proceedings of the International Conference on Computer Logic*, Lecture Notes in Computer Science 417, 1988.
11. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6. North-Holland, 1990.
12. P. Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65(2):525–549, 2000.
13. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1988.
14. R. Harper and J. Mitchell. Parametricity and variants of Girard's J operator. *Information Processing Letters*, 70:1–5, 1999.
15. J.-P. Jouannaud and M. Okada. Executable higher-order algebraic specification languages. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, 1991.
16. J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.
17. R. Matthes. *Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types*. PhD thesis, Ludwig Maximilians Universität, München, Germany, 1998.
18. C. McBride. *Dependently typed functional programs and their proofs*. PhD thesis, University of Edinburgh, United Kingdom, 1999.
19. N. P. Mendler. *Inductive Definition in Type Theory*. PhD thesis, Cornell University, United States, 1987.
20. C. Paulin-Mohring. Personal communication, 2001.
21. R. Pollack. Dependently typed records in type theory. *Formal Aspects of Computing*, 13(3–5):341–363, 2002.
22. Coq Development Team. *The Coq Proof Assistant Reference Manual – Version 7.3*. INRIA Rocquencourt, France, 2002. `http://coq.inria.fr/`.
23. B. Werner. *Une Théorie des Constructions Inductives*. PhD thesis, Université Paris VII, France, 1994.