Inductive Types as Equations

Lisa Allali[1] and Paul Brauner[2]

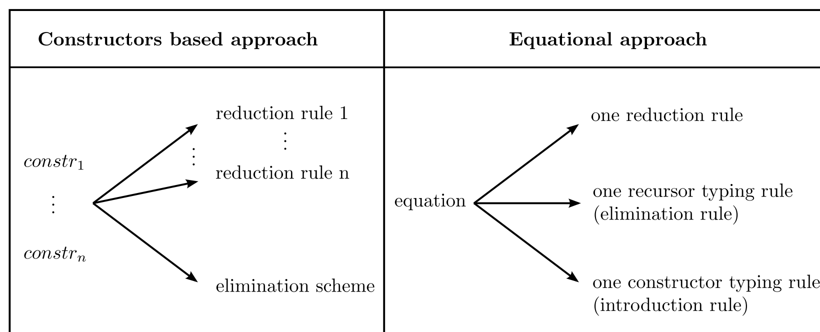
[1] École Polytechnique, INRIA & Région Ile de France allali@lix.polytechnique.fr
[2] INPL & LORIA paul.brauner@loria.fr

Abstract. Inductive types play a central role in proof assistants and programming languages. There are usually two ways of interpreting them in a Curry-Howard manner, either by encoding their inhabitants by lambda-terms (Church integers, Girard, Parigot), either by recursors (Gödel System T, Matin-Löf). We propose to link the two approaches the following way: recursors of inductive types are naturally defined as the elimination rules generated by the transformation of equations into supernatural deduction rules. This simplifies and breaks the asymmetry of usual inductive types presentations. Moreover, the strong normalization property of the resulting system is obtained by using a simple semantic argument.

1 Introduction

There are usually two ways of interpreting inductive types[1,2,3,4] in a Curry-Howard manner, either by encoding their inhabitants by lambda-terms (Church integers, Girard, Parigot), either by recursors (Gödel System T, Matin-Löf). We propose to link the two approaches.

An inductive type can be seen as a *definition*, as for instance the definition of Church integers in System F: $Nat = \forall X\ (X \Rightarrow (X \Rightarrow X) \Rightarrow X)$. This definition only allows iterations (the predecessor can not be reached in less than $n-1$ reduction steps). In order to get the predecessor in one step, the definition must be modified into an *equation*: $Nat = \forall X\ (X \Rightarrow (Nat \Rightarrow X \Rightarrow X) \Rightarrow X)$. In this paper, we give a central role to these equations in order to unify the two computational interpretations.



Instead of deducing the computational behaviour of the recursor from the types of the

constructors, we show how recursors and constructors are just *natural consequences* of the equation defining an inductive type. The ideal framework to observe this natural process is *superdeduction*. It is indeed once the equation is transformed into new inference rules in superdeduction that the computational behaviour of the recursor and the types of the constructors appear. This new approach has two major advantages:

- The system obtained is more symmetric:
  Usually, for one inductive type there are one recursor, n typing rules for constructors and n reduction rules. In our system, we naturally infer from the equation of an inductive type one introduction and one elimination rule for this inductive type. The introduction rule permits to type the constructors of the inductive type, the elimination rule types the recursor. There is only *one* reduction rule (instead of n in the case of an inductive type with n connectors). The usual constructors can be simulated by terms of the expected type.
- The strong normalization of this system is trivial thanks to the semantic method coming from deduction modulo called *super-consistency*[5].

2 Deduction Systems

2.1 Natural Deduction.

Our starting point is natural deduction for minimal predicate logic. The syntax of *terms* and *propositions* is given by

$$t = x \mid f(t, \ldots, t) \qquad A = P(t, \ldots, t) \mid (A \Rightarrow A) \mid \forall x\ A$$

We shall call *constants* zero-ary functions symbols and *predicate variables* zero-ary predicate symbols. We name *atoms* propositions with no connectives. The *proof-term* language is given by the following grammar, whose constructs are respectively typed by the usual typing rules (Ax), $(\Rightarrow I)$, $(\Rightarrow E)$, $(\forall I)$, $(\forall E)$ (see [6] for instance).

$$\pi ::= \alpha \mid \lambda\alpha.\pi \mid (\pi\ \pi') \mid \lambda x.\pi \mid (\pi\ t)$$

The variables $x, y, \ldots$ are variables of the first-order theory while $\alpha, \beta, \ldots$ are proof variables. As usual, the process of cut elimination is modeled by β-reduction.

Notation 1 (Vector) *We note $\overrightarrow{x}$ (potentially empty) variable sequences $(x_1, \ldots, x_n)$ and we write $(t\ \overrightarrow{x})$ for $t\ x_1\ \ldots\ x_n$. We similarly denote by $\overrightarrow{P}$ proposition sequences $(P_1, \ldots, P_n)$ and write $\overrightarrow{P} \Rightarrow A$ for $P_1 \Rightarrow \ldots \Rightarrow P_n \Rightarrow A$.*

2.2 Deduction Modulo (DM).

Deduction Modulo is a formalism that aims at distinguishing reasoning from computation in proofs by reasoning *modulo* some congruence. This may be explicited by a rephrasing of the inference rules, as shown for the implication elimination below.

$$(\Rightarrow E)\ \frac{\Gamma \vdash C \qquad \Gamma \vdash A}{\Gamma \vdash B}\ C \equiv A \Rightarrow B$$

The other rules of deduction modulo are build in the same way upon natural deduction (see figure 1 in appendix for the full system). The proof-terms are left unchanged w.r.t. natural deduction.

In this paper, we restrict ourselves to congruences expressed by the reflexive, symmetric and transitive closure of proposition rewrite systems defined as follows.

Definition 1 (Proposition rewrite system). *We call* proposition rewrite rule *every rule* $\mathsf{R} : P \to \varphi$ *rewriting atomic propositions* P *into arbitrary propositions* φ*. Moreover, we suppose that* $\mathcal{FV}(\varphi) \subseteq \mathcal{FV}(P)$*. We define a* proposition rewrite system *as an orthogonal, hence confluent set of proposition rewrite rules.*

Example 1 (Equality on naturals). Let us consider the rewrite system $\mathcal{R}$ formed by the proposition rewrite rule $\mathsf{R}_{\mathsf{eq}} : S(x) = S(y) \to x = y$. Then the type $(100 = 100) \Rightarrow (0 = 0)$ has $\lambda\alpha.\alpha$ as a proof in $DM_{\mathcal{R}}$ with only one step of reasoning but 100 steps of rewriting that are transparent in the proof:

$$\cfrac{(Ax)\ \cfrac{}{\alpha : 100 = 100 \vdash \alpha : 0 = 0}\ (100 = 100) \equiv (0 = 0)}{\vdash \lambda\alpha.\alpha : (100 = 100) \Rightarrow (0 = 0)}\ (\Rightarrow I)$$

While deduction modulo succeeds in hiding purely computational steps of reasoning, it fails at getting rid of the “noise” of trivial proof steps. Superdeduction [7] is a variation on the theme of Deduction Modulo, consisting at transforming a rewrite rule $P \to Q$ defining a predicate P into two new *inference rules*: one for the elimination and the other for the introduction of P. The notion of cuts is preserved yet enriched to fit the new deducing system: a cut in a proof is either the introduction followed by the elimination of the same connector, or the introduction followed by the elimination of the same *predicate*.

2.3 Supernatural Deduction (SND).

We start by illustrating supernatural deduction with the example of the rewrite rule defining the inclusion. The rewrite rule

$$\mathsf{R}_{\subseteq} : A \subseteq B \to \forall x\,(x \in A \Rightarrow x \in B)$$

is translated into the following inference rules.

$$(\mathsf{R}_{\subseteq} I)\ \frac{\Gamma, x \in A \vdash x \in B}{\Gamma \vdash A \subseteq B}\ x \notin \mathcal{FV}(\Gamma) \qquad (\mathsf{R}_{\subseteq} E)\ \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash t \in A}{\Gamma \vdash t \in B}$$

Let us see the formal definition of supernatural deduction. We call $SND^{\mathcal{R}}$ the supernatural deduction system associated to $\mathcal{R}$. The proof-terms are those of natural deduction extended with two constructs for each rewrite rule R of $\mathcal{R}$.

$$\pi ::= \mathsf{Cons}_{\mathsf{R}}(\overrightarrow{x}.\pi) \mid \mathsf{Rec}_{\mathsf{R}}(\pi, \overrightarrow{m}) \mid \ldots$$

In the pattern $\mathsf{Cons}_{\mathsf{R}}(\overrightarrow{x}.\pi)$, the vector $\overrightarrow{x}$ is a sequence of variables that may be either first-order or proof variables. They bind variables in π. In the term $\mathsf{Rec}_{\mathsf{R}}(\pi, \overrightarrow{m})$, the vector $\overrightarrow{m}$ is a sequence of terms that may be either first-order terms or proof-terms. We can now define the typing rules that correspond to the terms above. Consider a proposition rewrite rule R. Since we are working within the frame of minimal predicate logic, it has the following form.

$$\mathsf{R} : P \to \forall \overrightarrow{x_1}\ (Q_1 \Rightarrow \dots \forall \overrightarrow{x_n}\ (Q_n \Rightarrow \forall \overrightarrow{x}\ Q) \dots)$$

The following typing rules respectively introduce and eliminate the predicate P.

$$(\mathsf{R}I)\ \frac{\Gamma, \alpha_1 : Q_1, \dots, \alpha_n : Q_n \vdash \pi : Q}{\Gamma \vdash \mathsf{Cons}_{\mathsf{R}}(\overrightarrow{x_1}.\alpha_1.\dots.\overrightarrow{x_n}.\alpha_n.\overrightarrow{x}.\pi) : P}\ \overrightarrow{x_1} \dots, \overrightarrow{x_n}, \overrightarrow{x} \notin \mathcal{FV}(\Gamma)$$

$$(\mathsf{R}E)\ \frac{\Gamma \vdash \pi : P \quad \Gamma \vdash \pi_1 : Q_1 \quad \dots \quad \Gamma \vdash \pi_n : Q_n}{\Gamma \vdash \mathsf{Rec}_{\mathsf{R}}(\pi, \overrightarrow{t_1}, \pi_1, \dots, \overrightarrow{t_n}, \overrightarrow{u}, \pi_n)}$$

Where $\overrightarrow{u}$ (resp. $\overrightarrow{t_i}$) is a sequence of first-order terms of the same size as $\overrightarrow{x}$ (resp $\overrightarrow{x_i}$).

Example 2 (Proof-terms for the inclusion). Our definition of $\subseteq$ uses a witness and charges an assumption into the context. Thus, the associated proof-terms are those given by the following typing rules.

$$\frac{\Gamma, \alpha{:}(x \in X) \vdash \pi : (x \in Y)}{\Gamma \vdash \mathsf{Cons}_{\mathsf{R}_\subseteq}(x.\alpha.\pi) : (X \subseteq Y)}\ x \notin \mathcal{FV}(\Gamma) \quad \frac{\Gamma \vdash \pi : (X \subseteq Y) \qquad \Gamma \vdash \pi_1 : (t \in X)}{\Gamma \vdash \mathsf{Rec}_{\mathsf{R}_\subseteq}(\pi, t, \pi_1) : (t \in Y)}$$

Definition 2 (Generalized cut elimination). *The elimination of a generalized cut is represented by a reduction which transmits the witnesses and the lemmas to the proof.*

$$\mathsf{Rec}_{\mathsf{R}}(\mathsf{Cons}_{\mathsf{R}}(\overrightarrow{x}.\pi), \overrightarrow{m}) \rhd \pi\{\overline{x := m}\}$$

2.4 Supernatural Deduction Modulo (SNDM).

We finally define *supernatural deduction modulo*, which combines new inference rules *and* rewrite rules.

Definition 3 (Supernatural deduction modulo (SNDM)). *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two proposition rewrite systems. We call* supernatural deduction $\mathcal{R}_1$ modulo $\mathcal{R}_2$ *($SNDM^{\mathcal{R}_1}_{\mathcal{R}_2}$) the deduction system formed by $SND^{\mathcal{R}_1}$ where the propositions are considered modulo $\mathcal{R}_2$* after *the computation of the supernatural deduction rules. When $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) is empty we fall back in the case of deduction modulo (resp. supernatural deduction).*

Note that the cut elimination property of the system may be lost depending on the theory. For instance, rewrite rule $A \to A \Rightarrow A$ allows to type $(\lambda x.(x\ x)\ \lambda x.(x\ x))$ (resp. $\mathsf{Rec}(\mathsf{Cons}(\alpha.\mathsf{Rec}(\alpha, \alpha)), \mathsf{Cons}(\alpha.\mathsf{Rec}(\alpha, \alpha))))$ in *DM* (resp. *SND*). Some criteria concerning the rewrite system ensure strong normalization of *DM*, among them semantic ones as we shall see later. They transpose to *SNDM* thanks to the following.

Proposition 1 (Strong normalization property transfer). *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two proposition rewrite systems. Strong normalization of* $DM_{\mathcal{R}_1 \cup \mathcal{R}_2}$ *implies that of* $SNDM^{\mathcal{R}_2}_{\mathcal{R}_1}$.

Proof. We translate the $\mathsf{Rec}_{\mathsf{R}}(\mathsf{Cons}_{\mathsf{R}}(\overrightarrow{x}.\pi), \overrightarrow{m})$ redexes of *SNDM* by $\lambda \overrightarrow{x}.\pi\, \overrightarrow{m}$ ones in deduction modulo. The translation is well-typed thanks to the congruence of deduction modulo. Each reduction step in *SNDM* is either a ρ-reduction, either a β one. The reductions of supernatural deduction cuts are in finite number since they make the size of the proof decrease, and the β-reductions are trivially simulated by the translation.

3 Inductive Types

3.1 General idea

In this paper we propose to change the usual apprehension of inductive types and give a central role to the *equations* by showing how recursors and constructors are just *natural consequences* of the equation defining an inductive type. Moreover, it is easier to prove this system is strongly normalizing thanks to a semantic tool provided by deduction modulo called *super-consistency*[5].

The protocol to observe how the recursor and constructors are emerging from an equation and how we get the strong normalization is the following:

- We see the equation defining the inductive type (for instance $Nat = \forall X\ (X \Rightarrow (Nat \Rightarrow X \Rightarrow X) \Rightarrow X)$) as a rewrite rule ($Nat \rightarrow \forall X\ (X \Rightarrow (Nat \Rightarrow X \Rightarrow X) \Rightarrow X)$).
- We prove the deduction modulo system associated to this rewrite rule is strongly normalizing.
- We go from this modulo system to a superdeduction system by transforming the rewrite rule into two supernatural inference rules following the process explained in section 2.3.
- The superdeduction system resulting is strongly normalizing thanks to the strong normalization property transfer (proposition 1).
- In this new system we get an introduction and an elimination superrule for the inductive type defined by the original equation. The introduction rule permits to type the constructors of the inductive type, the elimination rule types the recursor. There is only *one* reduction rule (instead of n in the case of an inductive type with n connectors). This reduction rule allows to eliminate cuts beetween these two superrules.

3.2 Formal presentation

Definition 4 (Polarity). *Let X be a proposition variable, φ a proposition formed over atoms and $\Rightarrow$. We define "X* positive in *φ" and "X* negative in *φ" by mutual induction*

as follows.

$$X \text{ positive in } A \text{ if } A \text{ is atomic} \qquad X \text{ negative in } A \text{ if } A \text{ is atomic and } A \neq X$$

$$\frac{X \text{ positive in } \varphi}{X \text{ positive in } \forall x\ \varphi} \qquad \frac{X \text{ negative in } \varphi}{X \text{ negative in } \forall x\ \varphi}$$

$$\frac{X \text{ negative in } \varphi_1 \quad X \text{ positive in } \varphi_2}{X \text{ positive in } \varphi_1 \Rightarrow \varphi_2} \qquad \frac{X \text{ positive in } \varphi_1 \quad X \text{ negative in } \varphi_2}{X \text{ negative in } \varphi_1 \Rightarrow \varphi_2}$$

Example 3 (Positivity).

- $\mathbf{nat}$ is positive in $\tau \Rightarrow (\mathbf{nat} \Rightarrow \tau \Rightarrow \tau) \Rightarrow \tau$
- $\mathbf{list}$ is positive in $\tau \Rightarrow (\mathbf{nat} \Rightarrow \mathbf{list} \Rightarrow \tau \Rightarrow \tau) \Rightarrow \tau$
- $\mathbf{tree}$ is positive in $\tau \Rightarrow (\mathbf{tree} \Rightarrow \tau \Rightarrow \mathbf{tree} \Rightarrow \tau \Rightarrow \tau) \Rightarrow \tau$
- $\mathbf{ord}$ is positive in $\tau \Rightarrow (\mathbf{ord} \Rightarrow \tau \Rightarrow \tau) \Rightarrow ((\mathbf{nat} \Rightarrow \mathbf{ord}) \Rightarrow (\mathbf{nat} \Rightarrow \tau) \Rightarrow \tau) \Rightarrow \tau$

Definition 5 (Inductive Type). *Let X be a proposition variable, ϵ a unary predicate and τ a constant. X is an inductive type if it is defined by a rewrite rule of the form:*

$$\mathsf{R}_{\mathbf{X}} \ : \ X \rightarrow \forall \tau\ (P_1 \Rightarrow \ldots \Rightarrow P_n \Rightarrow \epsilon(\tau))$$

where every P_i is given by the following grammar.

$$P ::= \epsilon(\tau) \mid B \Rightarrow P \mid (\overrightarrow{A} \Rightarrow X) \Rightarrow (\overrightarrow{A} \Rightarrow \epsilon(\tau)) \Rightarrow P$$

$\overrightarrow{A}$ is an arbitrary predicate variable sequence, B is an arbitrary atomic predicate variable. X does not appear in them. The two occurrences of $\overrightarrow{A}$ denote the same sequence.

Example 4 (Inductive types).

$$\begin{array}{l} \mathsf{R}_{\mathbf{nat}} \ : \ \mathbf{nat} \rightarrow \epsilon(\tau) \Rightarrow (\mathbf{nat} \Rightarrow \epsilon(\tau) \Rightarrow \epsilon(\tau)) \Rightarrow \epsilon(\tau) \\ \mathsf{R}_{\mathbf{list}} \ : \ \mathbf{list} \rightarrow \epsilon(\tau) \Rightarrow (\mathbf{nat} \Rightarrow \mathbf{list} \Rightarrow \epsilon(\tau) \Rightarrow \epsilon(\tau)) \Rightarrow \epsilon(\tau) \\ \mathsf{R}_{\mathbf{tree}} : \mathbf{tree} \rightarrow \epsilon(\tau) \Rightarrow (\mathbf{tree} \Rightarrow \epsilon(\tau) \Rightarrow \mathbf{tree} \Rightarrow \epsilon(\tau) \Rightarrow \epsilon(\tau)) \Rightarrow \epsilon(\tau) \\ \mathsf{R}_{\mathbf{ord}} \ : \ \mathbf{ord} \rightarrow \epsilon(\tau) \Rightarrow (\mathbf{ord} \Rightarrow \epsilon(\tau) \Rightarrow \epsilon(\tau)) \\ \qquad\qquad \Rightarrow ((\mathbf{nat} \Rightarrow \mathbf{ord}) \Rightarrow (\mathbf{nat} \Rightarrow \epsilon(\tau)) \Rightarrow \epsilon(\tau)) \Rightarrow \epsilon(\tau) \end{array}$$

Lemma 1 (Positivity of inductive types). *Let X be an inductive type. The proposition variable X is positive in the right-hand side of its defining rewrite rule.*

Proof. Immediate. □

Let $X_1, \ldots, X_n$ be inductive types. To every proposition τ formed with $X_1, \ldots, X_n$ and $\Rightarrow$, we associate a constant $\dot{\tau}$. We call $\mathcal{R}_2$ the (infinite) system consisting of the decoding rules $\mathsf{R}_\epsilon : \epsilon(\dot{\tau}) \rightarrow \tau$. We call $\mathcal{R}_1$ the rewrite rules defining $X_1, \ldots, X_n$.

Theorem 1. *Deduction modulo for $\mathcal{R}_1 \cup \mathcal{R}_2$ ($DM_{\mathcal{R}_1 \cup \mathcal{R}_2}$) strongly normalizes.*

Proof. The proof relies on the fact that every X_i is positive in the right-hand side of its defining equation. This allows to build a fixpoint over the algebra of candidates as done in [8,9]. More generally, we can build a model of the rewrite system defining inductive types in every complete ordered pseudo-heyting algebra using a similar fixpoint argument, which entails strong normalization of the associated deduction modulo system as shown in [5]. The decoding rules are handled the following way: every $\dot{\tau}$ is interpreted by the denotation of τ in the alebra, and ϵ is interpreted by the identity. See [10] for an example. □

When translating the rewrite rules $X \longrightarrow \forall \tau\ (P_1 \Rightarrow \ldots \Rightarrow P_n \Rightarrow \epsilon(\tau))$ of $\mathcal{R}_2$ into supernatural deduction rules, we get a new type system which in turn enjoys the cut elimination property.

Corollary 1. *$SNDM^{\mathcal{R}_1}_{\mathcal{R}_2}$ strongly normalizes.*

Proof. By lemma 1. □

The general shape of these superrules is the following:

$$(\mathsf{R}_{\mathbf{X}} I)\ \frac{\Gamma, \alpha_1 : P_1, \ldots, \alpha_n : P_n \vdash \pi : \epsilon(\tau)}{\Gamma \vdash \mathsf{Cons}_{\mathsf{X}}(\tau.\alpha_1.\ldots.\alpha_n.\pi) : X}\ \tau \notin \mathcal{FV}(\Gamma)$$

$$(\mathsf{R}_{\mathbf{X}} E)\ \frac{\Gamma \vdash \pi : X \qquad \Gamma \vdash \pi_1 : P_1 \qquad \ldots \qquad \Gamma \vdash \pi_n : P_n}{\Gamma \vdash \mathsf{Rec}_{\mathsf{X}}(\pi, \tau, \pi_1, \ldots, \pi_n) : \epsilon(\tau)}$$

While the first rule allows to *construct* terms of type X, the second one *destructs* such terms by means of a recursor. Let us have a look at the elimination rule for $\mathbf{nat}$:

$$(\mathsf{R}_{\mathbf{nat}} E)\ \frac{\Gamma \vdash \pi : \mathbf{nat} \qquad \Gamma \vdash \pi_1 : \epsilon(\tau) \qquad \Gamma \vdash \pi_2 : \mathbf{nat} \Rightarrow \epsilon(\tau) \Rightarrow \epsilon(\tau)}{\Gamma \vdash \mathsf{Rec}_{\mathsf{nat}}(\pi, \tau, \pi_1, \pi_2) : \epsilon(\tau)}$$

We get the typing schema of the Gödel's system T recursor: given one type τ, a recursor instantiated for τ is built from a term π of type **nat**, a zero-ary function π_1 that returns a term of type τ (modulo $\mathcal{R}_1$) and a binary function π_2 that takes a natural number and the result of a recursive call as an argument and returns a term of type τ.

The introduction rule for $\mathbf{nat}$ is less usual. As outlined before, instead of two typing rules introducing $\mathsf{0}$ and S, we get one rule $\mathsf{R}_{\mathbf{nat}} I$ that allows to build every term of type $\mathbf{nat}$:

$$(\mathsf{R}_{\mathbf{nat}} I)\ \frac{\Gamma, \alpha : \mathbf{nat}, \beta : \mathbf{nat} \Rightarrow \epsilon(\tau) \Rightarrow \epsilon(\tau) \vdash \pi : \epsilon(\tau)}{\Gamma \vdash \mathsf{Cons}_{\mathsf{nat}}(\tau.\alpha.\beta.\pi) : \mathbf{nat}}\ \tau \notin \mathcal{FV}(\Gamma)$$

We therefore define a constructor as any function that returns a term built using an inductive type's introduction rule.

Definition 6 (Constructor). *Given an inductive type X, a* constructor *for X is a term $\mathsf{c} = \lambda \overrightarrow{x}.\mathsf{Cons}_{\mathsf{X}}(\tau.\alpha_1.\ldots.\alpha_n.\pi)$ of type $\overrightarrow{A} \Rightarrow X$. $\overrightarrow{A}$ is called the* arity *of* c.

Example 5. As defined below, $\mathsf{0}$ and S are constructors for **nat**.

$$\mathsf{0} = \mathsf{Cons_{nat}}(\tau.\alpha.\beta.\alpha)$$
$$\mathsf{S} = \lambda n.\mathsf{Cons_{nat}}(\tau.\alpha.\beta.\mathsf{Rec_{nat}}(n,\tau,\alpha,\beta))$$

Finally, the *only* reduction rule which eliminates $(\mathsf{R}_\mathbf{X}I)-(\mathsf{R}_\mathbf{X}E)$ cuts is:

$$\mathsf{Rec_X}(\mathsf{Cons_X}(\tau.\alpha_1.\ldots.\alpha_n.\pi),\dot{\tau},\pi_1,\ldots,\pi_2) \rhd \pi\{\tau := \dot{\tau},\alpha_1 := \pi_1,\ldots,\alpha_n := \pi_n\}$$

which, instantiated for $\mathbf{nat}$, gives the following.

$$\mathsf{Rec_{nat}}(\mathsf{Cons_{nat}}(\tau.\alpha.\beta.\pi),\tau,\pi_1,\pi_2) \rhd \pi\{\alpha := \pi_1,\beta := \pi_2\}$$

However, the constructor-specific computational behaviour which usually gives rise to several reduction rules is now contained in the constructor themselves. For instance, the particular constructors of the example above entail the well-known computational behaviour of system T. Notice that these are one-step reductions:

$$\mathsf{Rec_{nat}}(\mathsf{0},\tau,u,v) \rhd u$$
$$\mathsf{Rec_{nat}}((\mathsf{S}\ n),\tau,u,v) \rhd v\ n\ \mathsf{Rec_{nat}}(n,\tau,\alpha,\beta)$$

We are not restricted to these constructors thought. Indeed, the system allows for the definition of other terms of type $\mathbf{nat}$. Take for instance a sequence $u_0,\ldots,u_n$ of terms of type $\mathbf{nat}$. The term

$$\mathsf{Cons_{nat}}(\tau.\alpha.\beta.(\beta\ u_n\ (\ldots\ (\beta\ u_0\ \alpha)\ldots)))$$

has then type $\mathbf{nat}$ and represents the list $u_0,\ldots,u_n$. Constructors as they are commonly referred to are now a particular case of the notion of constructor entailed by the equation defining an inductive type. We call them *canonical constructors*.

3.3 Canonical Constructors

Although we believe that their symmetric presentation constitutes an elegant and sufficient definition of inductive types, with a substantial gain in the normalization proof, we still can recover the usual notion of constructor. Canonical constructors are derived from the equations defining recursive types. Given the rewrite rule $X \to \forall\tau\ (P_1 \Rightarrow \ldots \Rightarrow P_n \Rightarrow \epsilon(\tau))$, we get n constructors $\mathsf{c_1},\ldots,\mathsf{c_n}$.

We start by some preliminary definition.

Definition 7. *The* arity *of a proposition φ is a sequence of $\forall$ and $\Rightarrow$ symbols defined by induction on φ as follows*

- *if φ is atomic* $\mathrm{arity}(\varphi) = [\,]$,
- *if $\varphi = \varphi_1 \Rightarrow \varphi_2$ then* $\mathrm{arity}(\varphi) = (\Rightarrow,\mathrm{arity}(\varphi_2))$,
- *if $\varphi = \forall x\ \varphi_1$ then* $\mathrm{arity}(\varphi) = (\forall,\mathrm{arity}(\varphi_1))$.

Let φ be a proposition, a sequence for φ *is a sequence of distinct variables such that the n-th variable of the sequence is a proof variable if the n-th element of the arity of φ is $\Rightarrow$ and a term variable otherwise.*

Give a sequence $\overrightarrow{u}$ for $P_1, \ldots, P_n$, a variable τ, and $\overrightarrow{x}$ a sequence for P_i we now define the term $\theta^{\tau}_{\overrightarrow{u}}(\overrightarrow{x}, P_i, f)$ that intuitively corresponds to the computational behaviour of the reduction rule usually associated to P_i in classical presentations of inductive types. It is defined by induction on the derivation of P_i according to definition 5:

$$
\begin{aligned}
\theta^{\tau}_{\overrightarrow{u}}([\,], \epsilon(\tau), f) &= f \\
\theta^{\tau}_{\overrightarrow{u}}((x, \overrightarrow{x'}), B \Rightarrow P, f) &= \theta^{\tau}_{\overrightarrow{u}}(\overrightarrow{x'}, P, (f\ x)) \\
\theta^{\tau}_{\overrightarrow{u}}(x, \overrightarrow{x'}), (\overrightarrow{A} \Rightarrow X) \Rightarrow (\overrightarrow{A} \Rightarrow \epsilon(\tau)) \Rightarrow P, f) &= \theta^{\tau}_{\overrightarrow{u}}(\overrightarrow{x'}, P, (f\ x\ \Lambda(\tau, x, \overrightarrow{u})))
\end{aligned}
$$

where $\Lambda(\tau, x, \overrightarrow{u}) = \lambda \overrightarrow{y}.\mathsf{Rec}_{\mathsf{X}}((x\ \overrightarrow{y}), \tau, \overrightarrow{u})$ if $\overrightarrow{y}$ is a sequence for $\overrightarrow{A}$.

The canonical constructor associated to P_i is then defined the following way.

Definition 8 (Canonical constructor). *Let X be an inductive type defined by $X \to \forall\tau\ (P_1 \Rightarrow \ldots \Rightarrow P_n \Rightarrow \epsilon(\tau))$. Let $\overrightarrow{u} = u_1, \ldots, u_n$ be a sequence for $P_1, \ldots, P_n$. We define the proof-term*

$$\mathsf{C_i} = \lambda \overrightarrow{x}.(\mathsf{Cons}_{\mathsf{X}}(\tau.\overrightarrow{u}.\theta^{\tau}_{\overrightarrow{u}}(\overrightarrow{x}, P_i, u_i))$$

as the canonical constructor associated to P_i.

Example 6 (Ordinals). The canonical constructors associated to the definition of $\mathbf{ord}$ in the examples above are:

- $\mathsf{0} = \mathsf{Cons}_{\mathsf{ord}}(\tau.\alpha.\beta.\gamma.\alpha)$
- $\mathsf{S} = \lambda n.\mathsf{Cons}_{\mathsf{ord}}(\tau.\alpha.\beta.\gamma.\mathsf{Rec}_{\mathsf{ord}}(n, \tau, \alpha, \beta, \gamma))$
- $\mathsf{lim} = \lambda f.\mathsf{Cons}_{\mathsf{ord}}(\tau.\alpha.\beta.\gamma.(\lambda n.\mathsf{Rec}_{\mathsf{ord}}((f\ n), \tau, \alpha, \beta, \gamma)))$

4 Future Extensions

In this paper, we focus on simply typed recursors. Modern proof environments such as COQ or AGDA propose dependent type systems. The next logical step is thus to extend our approach to adapted frameworks like $\lambda\Pi$-modulo.

References

1. Werner, B.: Une Théorie des Constructions Inductives. PhD thesis, Université Paris 7 (1994)
2. Altenkirch, T.: Constructions, Inductive Types and Strong Normalization. PhD thesis, University of Edinburgh (November 1993)
3. Geuvers, H.: Inductive and coinductive types with iteration and recursion. Proceedings of the Workshop on Types for Proofs and Programs (November 1992)

4. Blanqui, F.: Inductive types in the calculus of algebraic constructions. TLCA'03 (2003) 61–86
5. Dowek, G.: Truth values algebras and proof normalization. In: TYPES. (2006) 110–124
6. Girard, J.Y.: Proofs and Types. Volume 7 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge (1989)
7. Brauner, P., Houtmann, C., Kirchner, C.: Principles of superdeduction. In: LICS '07: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science, Washington, DC, USA, IEEE Computer Society (2007) 41–50
8. Dowek, G., Werner, B.: Proof normalization modulo. Journal of Symbolic Logic **68**(4) (2003) 1289–1316
9. Dowek, G., Werner, B.: Arithmetic as a theory modulo. In Giesl, J., ed.: Proceedings of RTA'05. Volume 3467 of LNCS., Springer (2005) 423–437
10. Allali, L.: Algorithmic equality in Heyting arithmetic modulo. In: TYPES. (2007)

A Deduction modulo typing rules

$$(Ax)\ \frac{}{\Gamma \vdash \alpha : B}\ \alpha : A \in \Gamma \text{ and } A \equiv B$$

$$(\Rightarrow I)\ \frac{\Gamma \alpha : A \vdash \pi : B}{\Gamma \vdash \lambda \alpha . \pi : C}\ C \equiv (A \Rightarrow B)$$

$$(\Rightarrow E)\ \frac{\Gamma \vdash \pi : C \qquad \Gamma \vdash \pi' : A}{\Gamma \vdash (\pi\ \pi') : B}\ C \equiv (A \Rightarrow B)$$

$$(\forall I)\ \frac{\Gamma \vdash \pi : A}{\Gamma \vdash \lambda x\ \pi : B}\ B \equiv (\forall x\ A), x \notin FV(\Gamma)$$

$$(\forall E)\ \frac{\Gamma \vdash \pi : B}{\Gamma \vdash (\pi\ t) : A\{x := t\}}\ B \equiv (\forall x\ A)$$

Fig. 1. Typing rules for deduction modulo for the congruence $\equiv$