A semantic normalization proof for a system with recursors

Lisa Allali, Paul Brauner

A semantic normalization proof for a system with recursors

Lisa Allali
École Polytechnique & INRIA

Paul Brauner
INPL & LORIA

Abstract

Semantics methods have been used to prove cut elimination theorems for a long time. It is only recently that they have been extend to prove strong normalization results, in particular for theories in deduction modulo. However such semantic methods did not apply for systems with recursors like Gödel system T. We show in this paper how super natural deduction provides a bridge between superconsistency of arithmetic and strong normalization of system T. We then generalize this result to a family of inductive types before discussing the dependant case.

1. Introduction

When building a new theory, there are several criteria one wants this theory to meet. In particular the *cut elimination* property which means that the cut rule is redundant. This property ensures the consistency of the theory. In an intuitionistic framework (as it is the case in the present paper) it also gives the so-called *witness property*: if a proof is ending by the introduction of the existential quantifier one can exhibit a witness of this existence. *Normalization* of a theory is also a desirable property. It ensures the termination of the proof reduction process, gives a way to construct witnesses, and eases the potential implementation of the theory. In a system where all the cuts are captured by the reduction rules (like in the typed lambda calculus), cut elimination is a consequence of normalization.
Gentzen was the first to prove, with a syntactic method, the theorem of cut elimination in sequent calculus(Hauptsatz) [15]. More recently semantics approaches have been introduced (for instance in [26, 28, 25, 18]) for building proof with respect to some notion of model that has been used by De Marco and Lipton [9] to prove cut elimination of the Intuitionistic Higher-Order Logic, and by Okada [22] for intuitionistic Linear Logic (first and higher-order). The bright side of semantic proofs is that they abstract from synctactic details intrinsic to the type system and concentrate on essential arguments. Therefore, they obediently adapt to language extentions and allow to characterize classes of theories that meet cut elimination property.
More recently, a link between such semantic methods and *normalization* results has been done in the frame of *deduction modulo* [13]. This semantic criterion, called superconsistency, implies the normalization property of every theory expressed as a rewrite system. This has led for instance to the semantic proofs that Heyting's arithmetic (*i.e.* Peano axioms considered in an intuitionistic setting) is not only consistent, but also verifies the strong normalization property [1][14]. Although systems with recursors enjoy *syntactic* proofs of strong normalization (*cf* Tait's proof for Gödel system T [27]), they have been reluctant to this method so far.
Recursors are used to compute with inductive types on which today's proof-assistants heavily rely to express most theories. This is especially the case of the COQ proof assistant [29, 24, 16]. Also many recent programming languages with support for dependant types such as AGDA[21] include such recursors. Recursors allow the user to easily describe recursive functions and properties while ensuring the strong normalization of the system, which is proved (syntactically) once for all in the metatheory.

In this paper we propose to bridge the gap between deduction modulo and systems with recursors by exhibiting an intermediate system. This system relies on *supernatural deduction*, an instance of the superdeduction principle [5] for natural deduction. Supernatural deduction is yet another way to integrate a theory into natural deduction in the form of inferences rules and had initially been introduced in order to provide a typing system for the rewriting calculus [7]. Fortunately, system T computational behaviour is naturally captured (with no complexity speed-up) by some terms of this calculus whose typing rules exactly match those of the recursors. It is therefore sufficient to prove normalization of supernatural deduction for this particular theory to obtain that of system T. That's where semantic methods come into play.

Indeed, a precedent paper [4] has shown that deduction modulo and supernatural deduction are equivalently normalizing for the same theory. In other words: the way a theory is injected into a deduction system is immaterial for the normalization property. We shall then use

super-consistency arguments to ensure normalization of the supernatural deduction system simulating system T reductions. This is done by providing a family of so-called B-models for the theory from which this later supernatural deduction system has been derived.

The semantics arguments being clearly identified, we then generalize the whole reasoning to a system with recursors *à la* system T for a certain family of mutually inductive types. This confirms the validity of our approach and provides the foundations of a possible implementation (this system has actually already been played with in lemuridae, a proof-assistant prototype for the sequent calculus counterpart of supernatural deduction).

System T being rather a programming language than a deduction system, we finally discuss the extension of our approach to dependant types, *i.e.* predicate logic. Simulation through supernatural deduction is still valid. On the semantic side, the system T being in some way a impoverished version of arithmetic, its super-consistency [1] fits our needs. We show how our method adapts to deduction system with rewriting. The paper ends with a conjecture that a much larger family of inductive types could be captured by rewriting, as pioneered in [3].

2. Framework

2.1. Deduction systems

Predicate logic
Our starting point is natural deduction for predicate logic. The proof-term language is given by the following grammar, whose constructs are respectively typed by the usual typing rules (Ax), $(\Rightarrow I)$, $(\Rightarrow E)$, $(\wedge I)$, $(\wedge E_1)$, $(\wedge E_2)$, $(\vee I_1)$, $(\vee I_2)$, $(\vee E)$, $(\bot E)$, $(\forall I)$, $(\forall E)$, $(\exists I)$ and $(\exists E)$ (see [17] for instance).

$$\begin{array}{rl}\pi ::= & \alpha \\ & \mid \lambda\alpha.\pi \mid (\pi\ \pi') \\ & \mid \langle\pi,\pi'\rangle \mid \mathrm{fst}(\pi) \mid \mathrm{snd}(\pi) \\ & \mid i(\pi) \mid j(\pi) \mid (\delta\ \pi_1\ \alpha.\pi_2\ \beta.\pi_3) \\ & \mid I \mid (\delta_\bot\ \pi) \\ & \mid \lambda x.\pi \mid (\pi\ t) \\ & \mid \langle t,\pi\rangle \mid (\delta_\exists\ \pi\ x.\alpha.\pi')\end{array}$$

The variables $x, y, \ldots$ are variables of the first-order theory while $\alpha, \beta, \ldots$ are proof variables. Notice that the variables α,β and x are bound in $\lambda\alpha.\pi$, $\lambda x.\pi$, $(\delta\ \pi_1, \alpha.\pi_2, \beta.\pi_3)$ and $(\delta_\exists\ \pi, x.\alpha.\pi')$. As usual, the process of cut elimination is modeled by (generalized) β-reduction, whose rules are reminded below.

$$\begin{array}{rcl}(\lambda\alpha.\pi_1\ \pi_2) & \rhd & \pi_1\{\alpha := \pi_2\} \\ \mathrm{fst}(\langle\pi_1,\pi_2\rangle) & \rhd & \pi_1 \\ \mathrm{snd}(\langle\pi_1,\pi_2\rangle) & \rhd & \pi_2 \\ \delta\ (i(\pi_1))\ \alpha.\pi_2\ \beta.\pi_3 & \rhd & \pi_2\{\alpha := \pi_1\} \\ \delta\ (j(\pi_1))\ \alpha.\pi_2\ \beta.\pi_3 & \rhd & \pi_2\{\beta := \pi_1\} \\ (\lambda x.\pi\ t) & \rhd & \pi_1\{x := t\} \\ \delta_\exists\ \langle t,\pi_1\rangle\ x.\alpha.\pi' & \rhd & \pi_2\{x := t, \alpha := \pi_1\}\end{array}$$

Deduction Modulo (DM)
Deduction Modulo is a formalism that aims at distinguishing reasoning from computation in proofs. Modern type systems feature a rule so-called *conversion rule* which allows to identify propositions which are equal modulo beta-conversion.

$$(\textsc{Conv})\ \frac{\Gamma \vdash t : T \qquad T \equiv_\beta T'}{\Gamma \vdash t : T'}$$

A side-effect of this rule is to allow *computation* inside the proof, the computation being performed by proof reduction. The idea of deduction modulo [11] is to study the phenomenon of computation inside a proof in a simpler framework: predicate logic, where propositions are identified by a congruence. The congruence depends on the theory. It is usually defined as the symmetric and transitive closure of a rewrite system over first-order terms and propositions. Therefore, the typing rules of deduction modulo are those of natural deduction, modulo the 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 3 in appendix for the full system). In this paper, we restrict ourselves to theories expressed by proposition rewrite systems defined as follows. We call them *computational theories*.

Definition 1 (Proposition rewrite system). *We call* proposition rewrite rule *every rule* $\mathsf{R} : P \to \varphi$ *rewriting atomic propositions into propositions build upon the language of predicate logic restricted to the connectives* $\Rightarrow$ *and* $\forall$. *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.

We shall call $DM_\mathcal{R}$ the deduction modulo system parametrized by the rewrite system $\mathcal{R}$. The proof-terms are left unchanged *w.r.t* natural deduction, only the types are identified.

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

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

In deduction modulo, proof reduction may however not terminate, depending on the theory defined by $\mathcal{R}$. This is the case for very simple (and even consistent) theories like the one defined by $P \to (P \Rightarrow P)$ [13]. On the other hand, the strong normalization of $DM_{\mathcal{R}}$ implies the consistency of $\mathcal{R}$. Therefore, finding criteria which entails the strong normalization of deduction modulo is an active research topic, which has lead to elegant proofs of normalization for arithmetic and set theory [14, 1, 12].

Supernatural deduction (SND)

While deduction modulo succeeds in hiding purely computational steps of reasoning, it fails at getting rid of the "noise" of trivial proof steps. Supernatural deduction [19, 5] addresses this issue by turning a theory $\mathcal{R}$ into new inference rules for natural deduction. As an example, 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.

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

$$(\mathsf{R}_{\subseteq} E)\ \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash t \in A}{\Gamma \vdash t \in B}$$

The original idea of supernatural deduction is to propose a proof-term language that keeps trace of new rules during proof-reduction while raising the notion of *cut* to the level of predicates ($(\mathsf{R}_{\subseteq} I) - (\mathsf{R}_{\subseteq} E)$ cuts for the example above).

Let us see the formal definition of this system. 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:

$$\pi ::= \lambda\mathsf{R}(\overrightarrow{x}) \mid (\pi\ \mathsf{R}(\overrightarrow{m})) \mid \dots$$

They allow to interpret the super-rules. In the pattern $\mathsf{R}(\overrightarrow{x})$ the constructor R is applied to a sequence of variables that may be either term or proof variables. In the term $\mathsf{R}(\overrightarrow{m})$ it is applied to a sequence of terms that may be either terms of the theory or proof-terms. We can now define the typing rules that correspond to the terms above.

Definition 2. *The* arity *of a formula 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 formula, 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.

We note $\overrightarrow{x}$ variable sequences. Moreover for any variable t, $(t\ [\,]) = t$ and if $\overrightarrow{x} = (x, \overrightarrow{x'})$ then $(t\ \overrightarrow{x}) = ((t\ x)\ \overrightarrow{x'})$. If C has type ϕ and $\overrightarrow{x}$ is a sequence for ϕ, we may say that $\overrightarrow{x}$ is a sequence for C.

Definition 3 (Computation of the introduction rules). *Consider a proposition rewrite rule $\mathsf{R} : P \to \varphi$. Consider a sequence l for φ of variables that do not occur in the rule. We associate to R an introduction rule for P of the form*

$$(\mathsf{R}\ I)\ \frac{\mathrm{premI}(\Gamma, \varphi, l)}{\Gamma \vdash (\lambda\mathsf{R}(l).\pi) : P}\ \mathrm{cond}(\Gamma, \varphi, l)$$

Where the sequent $\mathrm{premI}(\Gamma, \varphi, l)$ and the condition $\mathrm{cond}(\Gamma, \varphi, l)$ are defined by induction on the structure of φ as follows

- *if φ is atomic then* $\begin{array}{l}\mathrm{premI}(\Gamma, \varphi, l) = (\Gamma \vdash \pi : \varphi)\\ \mathrm{cond}(\Gamma, \varphi, l)\ \ = \varnothing\end{array}$
- $\begin{array}{l}\mathrm{premI}(\Gamma, A \Rightarrow B, (\alpha, l)) = \mathrm{premI}((\Gamma, \alpha : A), B, l)\\ \mathrm{cond}(\Gamma, A \Rightarrow B, (\alpha, l))\ \ = \mathrm{cond}((\Gamma, A), B, l)\end{array}$
- $\begin{array}{l}\mathrm{premI}(\Gamma, \forall x\ A, (y, l)) = \mathrm{premI}(\Gamma, A\{x := y\}, l)\\ \mathrm{cond}(\Gamma, \forall x\ A, (y, l))\ \ = \mathrm{cond}(\Gamma, A\{x := y\}, l)\\ \qquad\qquad\qquad\qquad \cup \{y \notin \mathcal{FV}(\Gamma)\}\end{array}$

Definition 4 (Computation of the elimination rules). *Consider a proposition rewrite rule $\mathsf{R} : P \to \varphi$. Consider a sequence l for φ of names. We associate to R an elimination rule for P of the form*

$$(\mathsf{R}\ E)\ \frac{\Gamma \vdash \pi : P \qquad \mathrm{premE}(\Gamma, \varphi, l)}{\mathrm{concl}(\Gamma, (\pi\ \mathsf{R}(l)), \varphi, l)}$$

Where the multiset of sequents $\mathrm{premE}(\Gamma, \varphi, l)$ and the sequent $\mathrm{concl}(\Gamma, \pi', \varphi, l)$ are defined by induction on the structure of φ as follows

- *if φ is atomic then* $\begin{array}{l}\mathrm{concl}(\Gamma, \pi', \varphi, l) = (\Gamma \vdash \pi' : \varphi)\\ \mathrm{premE}(\Gamma, \varphi, l)\ \ = \varnothing\end{array}$
- $\begin{array}{l}\mathrm{concl}(\Gamma, \pi', A \Rightarrow B, (\tau, l)) = \mathrm{concl}(\Gamma, \pi', B, l)\\ \mathrm{premE}(\Gamma, A \Rightarrow B, (\tau, l))\ \ = \{\Gamma \vdash \tau : A\}\\ \qquad\qquad\qquad\qquad \cup \mathrm{premE}(\Gamma, B, l)\end{array}$
- $\begin{array}{l}\mathrm{concl}(\Gamma, \pi', \forall x\ A, (t, l)) = \mathrm{concl}(\Gamma, \pi', A\{x := t\}, l)\\ \mathrm{premE}(\Gamma, \forall x\ A, (t, l))\ \ = \mathrm{premE}(\Gamma, A\{x := t\}, l)\end{array}$

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:*

$$(\mathsf{R}_{\subseteq} I)\ \frac{\Gamma, \alpha{:}(x \in X) \vdash A : (x \in Y)}{\Gamma \vdash \lambda \mathsf{R}_{\subseteq}(x, \alpha).A : (X \subseteq Y)}\ x \notin \mathcal{FV}(\Gamma)$$

$$(\mathsf{R}_{\subseteq} E)\ \frac{\Gamma \vdash A : (X \subseteq Y) \qquad \Gamma \vdash B : (t \in X)}{\Gamma \vdash A\, \mathsf{R}_{\subseteq}(t, B) : (t \in Y)}$$

Definition 5 (Generalized cut elimination). *The elimination of a generalized cut is represented by a reduction which transmits the witnesses and the lemmas to the proof.*

$$(\lambda \mathsf{R}(\overrightarrow{x}).\pi\ \mathsf{R}(\overrightarrow{m})) \rhd_\rho \pi\{\overrightarrow{x := m}\}$$

When seeing supernatural deduction proof-terms as very simple ρ-terms of the rewriting calculus [8], the generalized cut elimination is then a ρ-reduction step. Hence the notation.

Supernatural deduction modulo (SNDM)
We finally define *supernatural deduction modulo*, which combines both systems.

Definition 6 (Supernatural deduction modulo). *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two rewrite systems composed of proposition rewrite rules. 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.*

Notice that we fall back in the case of deduction modulo (*resp.* supernatural deduction) when $\mathcal{R}_1$ (*resp.* $\mathcal{R}_2$) is empty. Let us now state the main result about supernatural deduction modulo.

Proposition 1 ($SNDM$ soundness and completeness). *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two proposition rewrite systems. $SNDM^{\mathcal{R}_1}_{\mathcal{R}_2}$ is sound and complete with respect to natural deduction within the theory formed by axioms of the form $\forall \overrightarrow{x}(P(\overrightarrow{x}) \Leftrightarrow \varphi(\overrightarrow{x}))$ for each proposition rewrite rule $P \to \varphi$ present in $\mathcal{R}_1 \cup \mathcal{R}_2$.*

Proof. By combining the similar results for deduction modulo and supernatural deduction respectively stated in [11] and [19]. □

Several criteria have been studied that ensure deduction modulo or supernatural deduction strong normalization. Transferring them from one system to another has been extensively studied in [4]. We adapt here one of the results that suits our needs.

Proposition 2 (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. The idea is to translate the $\lambda \mathsf{R}(\overrightarrow{x}).\pi\ \mathsf{R}(\overrightarrow{m})$ redexes of *SND* by $\lambda \overrightarrow{x}.\pi\ \overrightarrow{m}$ ones in deduction modulo which reduce to $\pi\{\overrightarrow{x := m}\}$ in a finite number of steps. The translation is well-typed thanks to the congruence of deduction modulo. □

2.2. Semantic tools

Let us see now a semantic criterion over computational theories that implies deduction modulo (thus supernatural deduction) strong normalization. It has been introduced in [10] and used in [1] to prove strong normalization of a computational presentation of Heyting's arithmetic.

Definition 7 ((full) Pseudo Heyting algebra). *Let B be a set and $\leq$ a relation on B. A structure $\langle B, \leq, \tilde{\wedge}, \tilde{\vee}, \tilde{\bot}, \tilde{\top}, \tilde{\forall}, \tilde{\exists}, \tilde{\Rightarrow} \rangle$ is a* Pseudo Heyting algebra *if forall x, y ,z, c in B and a in $\wp(B)$,*

- *$\leq$ is a preorder on B,*
- *$\tilde{\bot}$ is a minimum of B for $\leq$,*
- *$\tilde{\top}$ is a maximum of B for $\leq$,*
- *$x \tilde{\wedge} y$ is a lower bound of x and y,*
- *$x \tilde{\vee} y$ is a upper bound of x and y,*
- *$\tilde{\forall}$ and $\tilde{\exists}$ (infinite lower and upper bounds) are functions from $\wp(B)$ to B such that:*
- *$\tilde{\forall}$ (infinite lower bound) is a function from $\wp(B)$ to B such that:*
  - $x \in a \Rightarrow \tilde{\forall} a \leq x$,
  - $(\forall x \in a\ c \leq x) \Rightarrow c \leq \tilde{\forall} a$,
  - $x \in a \Rightarrow x \leq \tilde{\exists} a$,
  - $(\forall x \in a\ x \leq c) \Rightarrow \tilde{\exists} a \leq c$.
- $x \leq y \tilde{\Rightarrow} z \Leftrightarrow x \tilde{\wedge} y \leq z$.

Definition 8 (Ordered pseudo Heyting algebra). *An ordered pseudo Heyting algebra is a pseudo Heyting algebra together with a relation $\sqsubseteq$ on B such that*

- *$\sqsubseteq$ is an order relation,*
- *$\tilde{\top} \leq b$ and $b \sqsubseteq b'$ then $\tilde{\top} \leq b'$,*
- *$\tilde{\top}$ is a maximal element for $\sqsubseteq$ and $\tilde{\bot}$ is a minimal element for $\sqsubseteq$,*
- *$\tilde{\wedge}, \tilde{\vee}, \tilde{\forall}, \tilde{\exists}$ are monotonous, $\tilde{\Rightarrow}$ is left anti-monotonous and right monotonous.*

Definition 9 (Complete ordered PHA). *An ordered pseudo Heyting algebra is said to be* complete *if every subset of B has a greatest lower bound for $\sqsubseteq$.*

Definition 10 (Modulo intuitionistic model). *Let $\mathcal{L}$ be a language. An* Intuitionist model *$\mathcal{M}$ of $\mathcal{L}$ is :*

- *a set M,*
- *a complete ordered pseudo Heyting algebra B,*
- *for each function symbol f of arity n a function $\hat{f}$ from M^n to M,*
- *for each predicate symbol P of arity n a function $\hat{P}$ from M^n to B.*

Definition 11 (Denotation). *Let $\mathcal{M}$ be a model, A be a proposition and ϕ be an assignment. We define $[\![A]\!]_\phi$ as follows:*

$$\begin{array}{rcl}
[\![x]\!]_\phi & = & \phi(x) \\
[\![\bot]\!]_\phi & = & \tilde{\bot} \\
[\![\top]\!]_\phi & = & \tilde{\top} \\
[\![f(t_1,\ldots,t_n)]\!]_\phi & = & \hat{f}([\![t_1]\!]_\phi,\ldots,[\![t_n]\!]_\phi) \\
[\![P(t_1,\ldots,t_n)]\!]_\phi & = & \hat{P}([\![t_1]\!]_\phi,\ldots,[\![t_n]\!]_\phi) \\
[\![A \wedge B]\!]_\phi & = & [\![A]\!]_\phi \tilde{\wedge} [\![B]\!]_\phi \\
[\![A \vee B]\!]_\phi & = & [\![A]\!]_\phi \tilde{\vee} [\![B]\!]_\phi \\
[\![A \Rightarrow B]\!]_\phi & = & [\![A]\!]_\phi \tilde{\Rightarrow} [\![B]\!]_\phi \\
[\![\forall x\ A]\!]_\phi & = & \tilde{\forall}\{[\![A]\!]_{\phi,x:=v} \mid v \in M\} \\
[\![\exists x\ A]\!]_\phi & = & \tilde{\exists}\{[\![A]\!]_{\phi,x:=v} \mid v \in M\}
\end{array}$$

Definition 12 (Models for computational theory). *A* model of a computational theory *whose rewrite rules are $R_1 \to R'_1, \ldots, R_n \to R'_n$ is such that for each assignment ϕ, $[\![R_i]\!]_\phi = [\![R'_i]\!]_\phi$ for $1 \leq i \leq n$.*

Definition 13 (Super-consistency). *A computational theory is* super-consistent *if, for each compete ordered Heyting algebra $\mathcal{B}$, there exists a $\mathcal{B}$-model of this theory.*

Finally, the main property of a super-consistent theory is to bear a model valuated in the reducibility candidates algebra and thus to normalize [13].

Proposition 3 (Normalization). *If a computational theory $\mathcal{R}$ is super-consistent, then each proof in $DM_{\mathcal{R}_1}$ strongly normalizes.*

As a consequence, by proposition 2, if a computational theory $\mathcal{R}_1 \cup \mathcal{R}_2$ is super-consistent then $SNDM^{\mathcal{R}_2}_{\mathcal{R}_1}$ strongly normalizes.

3. Gödel System T

We introduce in this section a theory $DM_{\epsilon,\mathsf{nat}}$ and prove that its super-consistency implies the strong normalization of system T.

Definition 14. *The system T is a subset of natural deduction restricted to (Ax), $(\Rightarrow I)$ and $(\Rightarrow E)$. We add*

- *the constants* nat, $\mathsf{0}$, S *and a infinity of symbols* rec^τ *where τ is formed with* nat *and* $\Rightarrow$
- *the axioms* $\mathsf{0} : \mathsf{nat}$, $\mathsf{S} : \mathsf{nat} \Rightarrow \mathsf{nat}$ *and the axiom scheme* $\mathsf{rec}^\tau : \tau \Rightarrow (\mathsf{nat} \Rightarrow \tau \Rightarrow \tau) \Rightarrow \mathsf{nat} \Rightarrow \tau$
- *the reduction rules*

$$\begin{array}{rcl}
\lambda x.t\ u & \rhd & t\{x := u\} \\
\mathsf{rec}^\tau\ a\ f\ \mathsf{0} & \rhd & a \\
\mathsf{rec}^\tau\ a\ f\ (\mathsf{S}\ n) & \rhd & f\ n\ (\mathsf{rec}^\tau\ a\ f\ n)
\end{array}$$

Example 3 (Functions in System T).

$$\begin{array}{l}
+ = \lambda a{:}\mathsf{nat}.\lambda b{:}\mathsf{nat}.(\mathsf{rec}^{\mathsf{nat}}\ \mathsf{0}\ \lambda x{:}\mathsf{nat}.\lambda y{:}\mathsf{nat}.(\mathsf{S}\ y)\ b) \\
\times = \lambda a{:}\mathsf{nat}.\lambda b{:}\mathsf{nat}.(\mathsf{rec}^{\mathsf{nat}}\ \mathsf{0}\ \lambda x{:}\mathsf{nat}.\lambda y{:}\mathsf{nat}.(+\ y\ a)\ b)
\end{array}$$

The aim of this section is to provide an original semantic proof of system T normalization: we will first introduce a rewrite system defining a deduction modulo system $DM_{\epsilon,\mathsf{nat}}$. Then we show the super-consistency of $DM_{\epsilon,\mathsf{nat}}$ which implies its normalization. Finally we migrate from $DM_{\epsilon,\mathsf{nat}}$ to a supernatural deduction system $SNDM^{\mathsf{nat}}_\epsilon$ which faithfully mimics system T proofs and computational behavior. A final lemma reduces strong normalization for the system T to strong normalization for $SNDM^{\mathsf{nat}}_\epsilon$ and permits to conclude that system T is normalizing, as the process of migrating from deduction modulo to supernatural deduction preserves the strong normalization property (proposition 2).

3.1. The $DM_{\epsilon,\mathsf{nat}}$ system

Definition 15 ($DM_{\epsilon,\mathsf{nat}}$). *For any proposition P we can form with* nat *and* $\Rightarrow$ *we add a constant $\dot{p}$. We add a constant ϵ. The rewrite rules are:*

$$\begin{array}{l}
\epsilon(\dot{p}) \to P \\
\mathsf{nat} \to \forall p\ (\epsilon(p) \Rightarrow (\mathsf{nat} \Rightarrow \epsilon(p) \Rightarrow \epsilon(p)) \Rightarrow \mathsf{nat} \Rightarrow \epsilon(p))
\end{array}$$

Theorem 1. *The $DM_{\epsilon,\mathsf{nat}}$ system is super-consistent.*

Proof. Let B be an ordered and complete PHA. We build a B-model $\mathcal{M}$ of $DM_{\epsilon,\mathsf{nat}}$ as follows:
The domain M of $\mathcal{M}$ is B. $[\![\epsilon]\!] = id$.
We look for an interpretation of nat such that $[\![\mathsf{nat}]\!] = [\![\forall p\ (\epsilon(p) \Rightarrow (\mathsf{nat} \Rightarrow \epsilon(p) \Rightarrow \epsilon(p)) \Rightarrow \mathsf{nat} \Rightarrow \epsilon(p))]\!]$. For any element f of B, we build a model $\mathcal{M}_f$ where nat is interpreted by f. Let Φ be a function from B to B mapping f to $[\![\forall p\ (\epsilon(p) \Rightarrow (\mathsf{nat} \Rightarrow \epsilon(p) \Rightarrow \epsilon(p)) \Rightarrow \mathsf{nat} \Rightarrow \epsilon(p))]\!]^{\mathcal{M}_f}$. B is ordered and complete and Φ is monotone (because nat only appears in positive position in $\forall p\ (\epsilon(p) \Rightarrow (\mathsf{nat} \Rightarrow \epsilon(p) \Rightarrow \epsilon(p)) \Rightarrow \mathsf{nat} \Rightarrow \epsilon(p))$). Thus Φ has a fixpoint F. We chose this F to interpret nat in $\mathcal{M}$.
Finally for each $\dot{p}$ we chose : $\hat{\dot{p}} = [\![P]\!]^{\mathcal{M}}$.
By construction $\mathcal{M}$ is a B-model of $DM_{\epsilon,\mathsf{nat}}$. Thus $DM_{\epsilon,\mathsf{nat}}$ is super-consistent. □

Corollary 1. *$DM_{\epsilon,\mathsf{nat}}$ is strongly normalizing.*

Proof. By proposition 3. □

3.2. From $DM_{\epsilon,\mathsf{nat}}$ to $SNDM_\epsilon^{\mathsf{nat}}$

Definition 16 ($SNDM_\epsilon^{\mathsf{nat}}$). *We keep the rewrite rule scheme:*

$$\epsilon(\dot{p}) \to P$$

We replace the rewrite rule

$$\mathsf{nat} \to \forall p\ (\epsilon(p) \Rightarrow (\mathsf{nat} \Rightarrow \epsilon(p) \Rightarrow \epsilon(p)) \Rightarrow \mathsf{nat} \Rightarrow \epsilon(p))$$

by the inference rules of figure 1. The associated instance of the ρ reduction rule is :

$$\lambda \mathsf{R}_{\mathbf{nat}}(x,\alpha,\beta).\pi\ \mathsf{R}_{\mathbf{nat}}(t,\pi_0,\pi_s) \rhd_\rho \pi\{x,\alpha,\beta := t,\pi_0,\pi_s\}$$

Lemma 1. *$SNDM_\epsilon^{\mathsf{nat}}$ is normalizing.*

Proof. By proposition 2. □

Our goal is to deduce from this normalization theorem that system T is terminating. To do so, we need to prove that we can simulate the computational behaviour of the system T in $SNDM_\epsilon^{\mathsf{nat}}$, in particular that the two reduction rules of the re cursor are simulated in the only ρ rule. The constructors $\mathsf{0}$ and S are encoded by the following proof-terms in $SNDM_\epsilon^{\mathsf{nat}}$.

$$\begin{array}{rcl} \nu_0 & = & \lambda \mathsf{R}_{\mathbf{nat}}(p,\alpha,\beta).\alpha \\ \nu_\mathsf{S} & = & \lambda n.\lambda \mathsf{R}_{\mathbf{nat}}(p,\alpha,\beta). \\ & & (\beta\ n\ (\lambda x.\lambda y.\lambda z.(z\ \mathsf{R}_{\mathbf{nat}}(p,y,z)\ \alpha\ \beta\ n))) \end{array}$$

The reader may point up here that ν_S is not in normal form and could be have been reduced to $\lambda n.\lambda \mathsf{R}_{\mathbf{nat}}(p,\alpha,\beta).(\beta\ n\ (n\ \mathsf{R}_{\mathbf{nat}}(p,\alpha,\beta)))$. This is due to the fact that we have chosen to curry rec^τ, unlike other presentations of system T which impose that the recursor is always applied to three arguments at once. As a consequence, the β-expanded version of ν_S presented above eases the proof of lemma 3. We recognize then in ν_0 and ν_S the number 0 and the successor function defined on Parigot integers [23], which are a recursive (compared to iterative) version of Church integers.

Definition 17 (Translation from system T to $SNDM_\epsilon^{\mathsf{nat}}$).

$$\begin{array}{rcl} \|t\ u\| & = & \|t\|\ \|u\| \\ \|\lambda x.t\| & = & \lambda x.\|t\| \\ \|\mathsf{rec}^\tau_{\mathsf{nat}}\| & = & \lambda x.\lambda y.\lambda z.(z\ \mathsf{R}_{\mathbf{nat}}(\dot{\tau},x,y)) \\ \|\mathsf{O}\| & = & \nu_0 \\ \|\mathsf{S}\| & = & \nu_\mathsf{S} \\ \|x\| & = & x \quad \textit{if } x \textit{ is a variable} \end{array}$$

Lemma 2 (Soundness). *For all $\pi : T$ in System T then $\|\pi\| : T$ in $SNDM_\epsilon^{\mathsf{nat}}$.*

Proof. By induction on π. □

Lemma 3 (Simulation). *Let π and π' be two proofs in system T such that $\pi \rhd \pi'$, then $\|\pi\| \rhd_\rho^+ \|\pi'\|$ in SND_{nat}.*

Proof. By induction on π.
The non trivial cases are the following:.

- $\mathsf{rec}^\tau_{\mathsf{nat}}\ u\ v\ \mathsf{0} \rhd u$

$$\begin{array}{rcl} & & \|\mathsf{rec}^\tau_{\mathsf{nat}}\ u\ v\ \mathsf{0}\| \\ & = & \lambda x.\lambda y.\lambda z.(z\ \mathsf{R}_{\mathbf{nat}}(\dot{\tau},x,y))\ \|u\|\|v\|\ \nu_0 \\ \rhd_\rho^3 & & \nu_0\ \mathsf{R}_{\mathbf{nat}}(\dot{\tau},\|u\|,\|v\|) \\ & = & \lambda \mathsf{R}_{\mathbf{nat}}(p,\alpha,\beta).\alpha(\mathsf{R}_{\mathbf{nat}}(\dot{\tau},\|u\|,\|v\|)) \\ \rhd_\rho & & \|u\| \end{array}$$

- $\mathsf{rec}^\tau_{\mathsf{nat}}\ u\ v\ (\mathsf{S}\ n) \rhd v\ n\ (\mathsf{rec}^\tau_{\mathsf{nat}}\ u\ v\ n)$

$$\begin{array}{rcl} & & \|\mathsf{rec}^\tau_{\mathsf{nat}}\ u\ v\ (\mathsf{S}\ n)\| \\ = & & \lambda u.\lambda v.\lambda t.(t\ \mathsf{R}_{\mathbf{nat}}(\dot{\tau},u,v))\ \|u\|\ \|v\|\ (\nu_\mathsf{S}\ \|n\|) \\ \rhd_\rho^3 & & (\nu_\mathsf{S}\ \|n\|)\ \mathsf{R}_{\mathbf{nat}}(\dot{\tau},\|u\|,\|v\|) \\ = & & (\lambda n.\lambda \mathsf{R}_{\mathbf{nat}}(p,\alpha,\beta). \\ & & (\beta\ n\ (\lambda x.\lambda y.\lambda z.(z\ \mathsf{R}_{\mathbf{nat}}(p,y,z)\ \alpha\ \beta\ n)))\ \|n\|) \\ & & \mathsf{R}_{\mathbf{nat}}(\dot{\tau},\|u\|,\|v\|) \\ \rhd_\rho^2 & & \|v\|\ \|n\|\ (\lambda x.\lambda y.\lambda z.(z\ \mathsf{R}_{\mathbf{nat}}(\dot{\tau},y,z)\ \|u\|\ \|v\|\ \|n\|)) \\ = & & \|v\ n\ (\mathsf{rec}^\tau_{\mathsf{nat}}\ u\ v\ n)\| \end{array}$$

□

Theorem 2 (Relative normalization). *The strong normalization of $SNDM_\epsilon^{\mathsf{nat}}$ implies that of System T.*

Proof. Consider a reduction sequence $\pi \rhd \pi_1 \rhd \pi_2 \rhd \dots$ in system T. By lemma 3, $\|\pi\| \rhd_\rho^+ \|\pi_1\| \rhd_\rho^+ \|\pi_2\| \rhd_\rho^+ \dots$ is a reduction sequence in $SNDM_\epsilon^{\mathsf{nat}}$ and thus is finite according to lemma 1. So is the one in system T. □

4. Inductive types

We now generalize this result to a family of mutually inductive types. This opens the door to an implementation of a proof assistant with recursors but no primitive reduction rule for recursors.

4.1. Simple types with recursors

Let $\mathcal{TY}$ be a set of *type* symbols and $\mathcal{C}$ a set of *constructor* symbols:

$$\begin{array}{rcl} \mathcal{TY} & ::= & \mathbf{nat}, \mathbf{btree}, \mathbf{list}, \dots \\ \mathcal{C} & ::= & \mathsf{0}, \mathsf{S}, \mathsf{Cons}, \dots \end{array}$$

Definition 18 (Very strictly positive types). *Let $\mathcal{RT}$ be a subset of $\mathcal{TY}$. The set of very strictly positive types associated to $\mathcal{RT}$ is defined by:*

$$\mathcal{PT}(\mathcal{RT}) ::= \mathcal{RT} \mid \mathcal{RT} \Rightarrow \mathcal{PT}(\mathcal{RT})$$

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

$$(\mathsf{R}_{\mathbf{nat}} E)\ \frac{\Gamma \vdash \pi : \mathsf{nat} \qquad \Gamma \vdash \pi_0 : \epsilon(t) \qquad \Gamma \vdash \pi_s : \mathsf{nat} \Rightarrow \epsilon(t) \Rightarrow \epsilon(t)}{\Gamma \vdash \pi\ \mathsf{R}_{\mathbf{nat}}(t, \pi_0, \pi_s) : \epsilon(t)}$$

Figure 1. Typing rules of SND_{nat}

Definition 19 (Signature). *A signature Σ is a set of pairs of typed constructors $(\mathsf{C_i} : P_i) \in \mathcal{C} \times \mathcal{PT}(\mathcal{RT})$ where $\mathcal{RT} \subseteq \mathcal{TY}$. We say that $\mathsf{C_i}$ is a* constructor for $\mathbf{rt}$ *if the conclusion type of P_i is $\mathbf{rt}$,* i.e. *$P_i = \ldots \Rightarrow \mathbf{rt}$.*

Example 4 (Trees and Forests). *In the signature Σ constituted by the follwing set:*

$$\begin{array}{rcl} \mathsf{Node} & : & \mathbf{frst} \Rightarrow \mathbf{tree} \\ \mathsf{Leaf} & : & \mathbf{frst} \\ \mathsf{Cons} & : & \mathbf{tree} \Rightarrow \mathbf{frst} \Rightarrow \mathbf{frst} \end{array}$$

Node is a constructor for $\mathbf{tree}$, Leaf and Cons are constructors for $\mathbf{frst}$.

From here on let Σ be a signature for a subset $\mathcal{RT}$ of $\mathcal{TY}$. We call $\mathcal{ST}$ (simple types) the set of propositions formed by $\mathcal{RT}$ and $\Rightarrow$. We call ML (minimal propositional logic) the subset of natural deduction restricted to (Ax), $(\Rightarrow I)$ and $(\Rightarrow E)$. In particular the proof-terms are restricted to variables, abstractions and applications.

For every $\mathbf{rt} \in \mathcal{RT}$, $\tau \in \mathcal{ST}$ and $\varphi \in \mathcal{PT}$ we define a proposition $\Delta^{\tau}_{\mathbf{rt}}(\varphi)$ by induction on φ as follows.

$$\begin{array}{lcll} \Delta^{\tau}_{\mathbf{rt}}(\mathbf{A}) & = & \tau & \text{if } \mathbf{A} \in \mathcal{RT} \\ \Delta^{\tau}_{\mathbf{rt}}(\mathbf{A} \Rightarrow B) & = & \mathbf{A} \Rightarrow \Delta^{\tau}_{\mathbf{rt}}(B) & \text{if } \mathbf{A} \neq \mathbf{rt} \\ \Delta^{\tau}_{\mathbf{rt}}(\mathbf{rt} \Rightarrow B) & = & \mathbf{rt} \Rightarrow \tau \Rightarrow \Delta^{\tau}_{\mathbf{rt}}(B) & \end{array}$$

This intuitively corresponds to the part of a constructor in the elimination scheme we will associate to $\mathbf{rt}$. As an example, $\Delta^{\tau}_{\mathbf{nat}}(\mathbf{nat} \Rightarrow \mathbf{nat}) = \mathbf{nat} \Rightarrow \tau \Rightarrow \tau$.

Definition 20 (Elimination scheme). *To every recursive type $\mathbf{rt} \in \mathcal{RT}$ we associate an elimination scheme ε^{τ} parametrized by $\tau \in \mathcal{ST}$:*

$$\varepsilon^{\tau}(\mathbf{rt}) = \Delta^{\tau}_{\mathbf{rt}}(t_1) \Rightarrow \ldots \Rightarrow \Delta^{\tau}_{\mathbf{rt}}(t_n) \Rightarrow \mathbf{rt} \Rightarrow \tau$$

where $t_1, \ldots, t_n = \{P_i \mid (\mathsf{C_i} : P_i) \in \Sigma\}$

We then enrich the deduction system with respect to Σ by adding an axiom for each constructor of Σ as well as recursor constants for each recursive type typed by the corresponding elimination scheme.

Definition 21 (Simple type system with recursors).

$$ML_{\Sigma} = ML \cup \Sigma \cup \{\mathsf{rec}^{\tau}_{\mathbf{rt}} : \varepsilon^{\tau}(\mathbf{rt}) \mid \mathbf{rt} \in \mathcal{RT}\}$$

Example 5 (Trees and Forests). *Given the signature Σ of the previous example, ML_{Σ} is the type system ML enriched with the axioms* $\mathsf{Node} : \mathbf{frst} \Rightarrow \mathbf{tree}$, $\mathsf{Leaf} : \mathbf{frst}$, $\mathsf{Cons} : \mathbf{tree} \Rightarrow \mathbf{frst} \Rightarrow \mathbf{frst}$ *as well as the two axiom schemes parametrized by τ:*

$$\begin{array}{rcl} \mathsf{rec}^{\tau}_{\mathbf{tree}} & : & \mathbf{frst} \Rightarrow \mathbf{tree} \Rightarrow \tau \\ \mathsf{rec}^{\tau}_{\mathbf{frst}} & : & \tau \Rightarrow (\mathbf{tree} \Rightarrow \mathbf{frst} \Rightarrow \tau \Rightarrow \tau) \Rightarrow \mathbf{frst} \Rightarrow \tau \end{array}$$

4.1.1 Reduction rules

Let τ be some type. Let $\mathbf{rt} \in \mathcal{RT}$, $(\mathsf{C} : P)$ be one of his constructors, $(\overrightarrow{u}, t)$ a sequence for $\mathsf{rec}^{\tau}_{\mathbf{rt}}$, $\overrightarrow{x}$ a sequence for C, and f a variable. We define the term $\Theta^{\tau}_{\mathbf{rt},\overrightarrow{u}}(\overrightarrow{x}, P, f)$ by induction on both $\overrightarrow{x}$ and P as follows.

$$\begin{array}{l} \Theta^{\tau}_{\mathbf{rt},\overrightarrow{u}}([\,], \mathbf{rt}, f) = f \\ \Theta^{\tau}_{\mathbf{rt},\overrightarrow{u}}((x,\overrightarrow{x'}), \mathbf{A} \Rightarrow B, f) = \Theta^{\tau}_{\mathbf{rt},\overrightarrow{u}}(\overrightarrow{x'}, B, (f\ x)) \text{ if } \mathbf{A} \neq \mathbf{rt} \\ \Theta^{\tau}_{\mathbf{rt},\overrightarrow{u}}((x,\overrightarrow{x'}), \mathbf{rt} \Rightarrow B, f) = \Theta^{\tau}_{\mathbf{rt},\overrightarrow{u}}(\overrightarrow{x'}, B, (f\ x\ (\mathsf{rec}^{\tau}_{\mathbf{rt}}\ \overrightarrow{u}\ x))) \end{array}$$

This corresponds to the right hand-side of the reduction rule we will associate to a constructor of $\mathbf{rt}$. As an example, we get the following term for the type of the constructor S.

$$\Theta^{\tau}_{\mathbf{nat},(a,f)}((n), \mathbf{nat} \Rightarrow \mathbf{nat}, f) = f\ n\ (\mathsf{rec}^{\tau}_{\mathbf{nat}}\ a\ f\ n)$$

Definition 22 (Reduction rules associated to a type). *Let τ be some type. Let $\mathbf{rt} \in \mathcal{RT}$. We name $\mathcal{R}^{\tau}_{\mathbf{rt}}$ the set of reduction rules composed of*

$$(\mathsf{rec}^{\tau}_{\mathbf{rt}}\ \overrightarrow{u}\ (\mathsf{C_i}\ \overrightarrow{x_i})) \rhd \Theta^{\tau}_{\mathbf{rt},\overrightarrow{u}}(\overrightarrow{x_i}, P_i, u_i)$$

for every $(\mathsf{C_i} : P_i)$ constructor of $\mathbf{rt}$, where $\overrightarrow{x_i}$ is a sequence for $\mathsf{C_i}$ and $\overrightarrow{u}$ is a sequence $u_1, \ldots, u_n$ of variables such that for all variable t, $(\overrightarrow{u}, t)$ is a sequence for $\mathsf{rec}^{\tau}_{\mathbf{rt}}$.

The set $\mathcal{R}(\Sigma)$ of reduction rules scheme parametrized by τ associated to a signature Σ is then naturally defined as:

$$\mathcal{R}(\Sigma) = \bigcup_{\mathbf{rt} \in \mathcal{RT}} \mathcal{R}^{\tau}_{\mathbf{rt}}$$

Example 6 (Trees and Forests). *For the signature Σ proposed in example 5, we get the following reduction system.*

$$\begin{array}{lcl} \mathsf{rec}^{\tau}_{\mathbf{tree}}\ u_1\ (\mathsf{Node}\ t_1) & \rhd & u_1\ t_1 \\ \mathsf{rec}^{\tau}_{\mathbf{frst}}\ u_1\ u_2\ \mathsf{Leaf} & \rhd & u_1 \\ \mathsf{rec}^{\tau}_{\mathbf{frst}}\ u_1\ u_2\ (\mathsf{Cons}\ t_1\ t_2) & \rhd & u_2\ t_1\ t_2\ (\mathsf{rec}^{\tau}_{\mathbf{frst}}\ u_1\ u_2\ t_2) \end{array}$$

Definition 23 (ML_Σ proof reduction). *The proof-terms of ML_Σ are identified by the union of $\mathcal{R}(\Sigma)$ and β-reduction.*

Proposition 4 (Subject reduction). *For all signature Σ, ML_Σ enjoys the subject reduction property.*

Proof. The rules of $\mathcal{R}(\Sigma)$ are type preserving. □

4.2. Translation to *SNDM*

We will now encode both ML_Σ judgments and computational behaviour into the $SNDM^{\mathcal{R}_2}_{\mathcal{R}_1}$ supernatural deduction modulo system. The predicate symbols are those of $\mathcal{RT}$ along with a unary predicate ϵ. To each proposition $\tau \in \mathcal{ST}$ we associate a constant symbol $\dot{\tau}$ of sort κ. We finally define $\mathcal{R}_1$ as the infinite proposition rewrite system which reifies them to the propositional level:

$$\mathcal{R}_1 = \{\epsilon(\dot{\tau}) \to \tau \mid \tau \in \mathcal{ST}\}$$

Remark 1. *The infinite nature of $\mathcal{R}_1$ is not intrinsic of our encoding. We could actually have used a finite one consisting in one constant $\dot{\mathbf{rt}}$ per recursive type symbol and its a decoding rule $\epsilon(\dot{\mathbf{rt}}) \to \mathbf{rt}$, along with a binary predicate symbol $\dot{\Rightarrow}$ decoded by the rule $\dot{\Rightarrow}(x,y) \to x \Rightarrow y$. As an example, the proposition $\dot{\Rightarrow}(\dot{\Rightarrow}(\dot{\mathbf{nat}}, \dot{\mathbf{nat}}), \dot{\mathbf{nat}})$ would be congruent to $(\mathbf{nat} \Rightarrow \mathbf{nat}) \Rightarrow \mathbf{nat}$ in this setting. However, for the sake of simplicity, we stick to the infinite version in this paper.*

Let us now define $\mathcal{R}_2$ according to the signature Σ. For every $\mathbf{rt} \in \mathcal{RT}, \varphi \in \mathcal{PT}$ and first-order variable p, we define a proposition $\delta^t_{\mathbf{rt}}(\varphi)$ by induction on the structure of φ as follows.

$$\begin{array}{lcll} \delta^p_{\mathbf{rt}}(\mathbf{A}) & = & \epsilon(p) & \text{if } \mathbf{A} \in \mathcal{RT} \\ \delta^p_{\mathbf{rt}}(\mathbf{A} \Rightarrow B) & = & \mathbf{A} \Rightarrow \delta^p_{\mathbf{rt}}(B) & \text{if } \mathbf{A} \neq \mathbf{rt} \\ \delta^p_{\mathbf{rt}}(\mathbf{rt} \Rightarrow B) & = & \mathbf{rt} \Rightarrow \epsilon(p) \Rightarrow \delta^p_{\mathbf{rt}}(B) & \end{array}$$

Definition 24 (Proposition rewrite rules associated to rt). *To every recursive type $\mathbf{rt} \in \mathcal{RT}$ we associate proposition rewrite rule $\mathsf{R}_{\mathbf{rt}}$ parametrized by $\tau \in \mathcal{ST}$:*

$$\mathsf{R}_{\mathbf{rt}} : \mathbf{rt} \to \forall p\, \delta^p_{\mathbf{rt}}(t_1) \Rightarrow \ldots \Rightarrow \delta^p_{\mathbf{rt}}(t_n) \Rightarrow \epsilon(p)$$

where $t_1, \ldots, t_n = \{P_i \mid (\mathsf{C_i} : P_i) \in \Sigma\}$

Let us state the essential property of these rules.

Lemma 4. *Positivity For every $\mathbf{rt}$ type in $\mathcal{RT}$ and $\mathsf{R}_{\mathbf{rt}} : \mathbf{rt} \to \phi$ the associated rewrite rule, the occurrences of $\mathbf{rt}$ are in positive position in ϕ.*

Proof. By induction on the type of $\mathbf{rt}$ constructors. □

These rules constitute $\mathcal{R}_2$:

$$\mathcal{R}_2 = \{\mathsf{R}_{\mathbf{rt}} \mid \mathbf{rt} \in \Sigma\}$$

which means that for each $\mathbf{rt} \in \mathcal{RT}$ we get two inference rules $(\mathbf{rt}I)$ and $(\mathbf{rt}E)$ whose general shape is given by figure 2.

Let us see now how we encode the constructors of ML_Σ with proof-terms of $SNDM^{\mathcal{R}_2}_{\mathcal{R}_1}$. Let $(\mathsf{C} : P)$ be a constructor of $\mathbf{rt} \in \mathcal{RT}$, $(p, \overrightarrow{u})$ a sequence for $\mathsf{R}_{\mathbf{rt}}$, $\overrightarrow{x}$ a sequence for C, and f a proof variable. We define the term $\theta^p_{\mathbf{rt},\overrightarrow{u}}(\overrightarrow{x}, P, f)$ by induction on both $\overrightarrow{x}$ and P as follows.

$$\begin{array}{ll} \theta^p_{\mathbf{rt},\overrightarrow{u}}([\,], \mathbf{rt}, f) & = f \\ \theta^p_{\mathbf{rt},\overrightarrow{u}}((x,\overrightarrow{x'}), \mathbf{A} \Rightarrow B, f) & = \theta^p_{\mathbf{rt},\overrightarrow{u}}(\overrightarrow{x'}, B, (f\ x)) \text{ if } \mathbf{A} \neq \mathbf{rt} \\ \theta^p_{\mathbf{rt},\overrightarrow{u}}((x,\overrightarrow{x'}), \mathbf{rt} \Rightarrow B, f) & = \theta^p_{\mathbf{rt},\overrightarrow{u}}(\overrightarrow{x'}, B, (f\ x\ r(p, x, \overrightarrow{u}))) \end{array}$$

Where $r(p, x, \overrightarrow{u}) = \lambda \overrightarrow{v}.\lambda y.(y\ \mathsf{R}_{\mathbf{rt}}(p, \overrightarrow{v})\ \overrightarrow{u}\ x)$

Definition 25 (Proof-term encoding a constructor). *Let $(\mathsf{C_i} : P_i)$ be a constructor of $\mathbf{rt} \in \mathcal{RT}$, $\overrightarrow{x}$ a sequence for $\mathsf{C_i}$ and $(p, \overrightarrow{u}) = p, u_1, \ldots, u_n$ a sequence for $\mathsf{R}_{\mathbf{rt}}$. We define the proof-term*

$$\nu_{\mathsf{C_i}} = \lambda \overrightarrow{x}.\lambda \mathsf{R}_{\mathbf{rt}}(p, \overrightarrow{u}).\theta^p_{\mathbf{rt},\overrightarrow{u}}(\overrightarrow{x}, P_i, u_i)$$

as the proof-term encoding the constructor $\mathsf{C_i}$.

Example 7 (Trees and Forests). *The proposition rewrite rules $\mathsf{R}_{\mathbf{tree}}$ and $\mathsf{R}_{\mathbf{frst}}$ respectively associated to $\mathbf{tree}$ and $\mathbf{forest}$ are:*

$$\begin{array}{l} \mathsf{rec}^\tau_{\mathbf{tree}} \to \forall p\, \mathbf{frst} \Rightarrow \epsilon(p) \\ \mathsf{rec}^\tau_{\mathbf{frst}} \to \forall p\, \epsilon(p) \Rightarrow (\mathbf{tree} \Rightarrow \mathbf{frst} \Rightarrow \epsilon(p) \Rightarrow \epsilon(p)) \Rightarrow \epsilon(p) \end{array}$$

The constructors Node, Leaf *and* Cons *are encoded by the following proof-terms in supernatural deduction.*

$$\begin{array}{lcl} \nu_{\mathsf{Node}} & = & \lambda x.\lambda \mathsf{R}_{\mathbf{tree}}(p, \alpha).(\alpha\ x) \\ \nu_{\mathsf{Leaf}} & = & \lambda \mathsf{R}_{\mathbf{frst}}(p, \alpha, \beta).\alpha \\ \nu_{\mathsf{Cons}} & = & \lambda x.\lambda y.\lambda \mathsf{R}_{\mathbf{frst}}(p, \alpha, \beta). \\ & & (\beta\ x\ y\ (\lambda\delta.\lambda\gamma.\lambda z.(z\ \mathsf{R}_{\mathbf{frst}}(p, \delta, \gamma))\ \alpha\ \beta\ y)) \end{array}$$

Definition 26 (Translation from ML_Σ to $SNDM^{\mathcal{R}_2}_{\mathcal{R}_1}$).

$$\begin{array}{rcl} \|t\ u\| & = & \|t\|\ \|u\| \\ \|\lambda x.t\| & = & \lambda x.\|t\| \\ \|\mathsf{rec}^\tau_{\mathbf{rt}}\| & = & \lambda \overrightarrow{x}.\lambda t.(t\ \mathsf{R}_{\mathbf{rt}}(\dot{\tau}, \overrightarrow{x})) \\ \|\mathsf{C}\| & = & \nu_{\mathsf{C}} \\ \|x\| & = & x \quad \textit{if } x \textit{ is a variable} \end{array}$$

Lemma 5 (Soundness). *For all proof-term π in ML_Σ, if $\pi : T$ then $\|\pi\| : T$ in $SNDM^{\mathcal{R}_2}_{\mathcal{R}_1}$.*

Proof. By induction on π. □

$$(\mathbf{rt}I)\ \frac{\Gamma, \alpha_1 : \delta^p_{\mathbf{rt}}(t_1), \ldots\ \alpha_n : \delta^p_{\mathbf{rt}}(t_n) \vdash \pi : \epsilon(p)}{\Gamma \vdash \lambda \mathsf{R}_{\mathbf{rt}}(x, \alpha_1, \ldots, \alpha_n).\pi : \mathbf{rt}}\ p \notin \mathcal{FV}(\Gamma)$$

$$(\mathbf{rt}E)\ \frac{\Gamma \vdash \pi : \mathbf{rt} \qquad \Gamma \vdash \pi_0 : \delta^p_{\mathbf{rt}}(t_1) \qquad \ldots \qquad \Gamma \vdash \pi_n : \delta^p_{\mathbf{rt}}(t_n)}{\Gamma \vdash \pi\ \mathsf{R}_{\mathbf{rt}}(t, \pi_0, \ldots, \pi_s) : \epsilon(t)}$$

Figure 2. General shape of $SNDM^{\mathcal{R}_2}_{\mathcal{R}_1}$ typing rules

Theorem 3 (Simulation). *Let π and π' be two proofs in ML_Σ such that $\pi \rhd \pi'$, then $\|\pi\| \rhd^+_\rho \|\pi'\|$.*

Proof. The interesting case is that of $\mathsf{rec}^\tau_{\mathbf{rt}}\ \overrightarrow{u}\ t$ which is treated by induction on the constructors of $\mathbf{rt}$. □

Corollary 2 (Relative normalization). *The strong-normalization of $SND^{\mathcal{R}_2}_{\mathcal{R}_1}$ implies that of ML_Σ.*

4.3. Strong normalization

We shall now prove the super-consistency of $DM_{\mathcal{R}_1 \cup \mathcal{R}_2}$, therefore strong normalization of $SNDM^{\mathcal{R}_2}_{\mathcal{R}_1}$.

Theorem 4. *$\mathcal{R}_1 \cup \mathcal{R}_2$ is superconsistent.*

Proof. Let B be an ordered and complete PHA. We build a B-model $\mathcal{M}$ of $\mathcal{R}_1 \cup \mathcal{R}_2$ as follows. The domain M of $\mathcal{M}$ is B. We therefore interpret ϵ by the identity on B.
To give an interpretation to the zero-ary predicates $\mathbf{rt_1}, \ldots, \mathbf{rt_n}$ of $\mathcal{RT}$, the proof slightly differs from the one for System T since we have to handle mutually recursive types (trees and forests for instance). For every tuple $(f_1, \ldots, f_n)$ of B^n we can define a model $\mathcal{M}_{(f_1,\ldots,f_n)}$ of the language $\forall, \Rightarrow, \mathbf{rt_1}, \ldots, \mathbf{rt_n}$ where $\mathbf{rt_1}$ is interpreted by f_1, $\mathbf{rt_2}$ by f_2, etc. We call $\mathcal{E}_\mathcal{M}$ the set made of of these models for every $(f_1, \ldots, f_n)$ of B^n. We then define a function Φ from $\mathcal{E}_\mathcal{M}$ to $\mathcal{E}_\mathcal{M}$ which maps a model $\mathcal{M}_{(f_1,\ldots,f_n)}$ to a model $\mathcal{M}_{(f'_1,\ldots,f'_n)}$ where f'_i is the interpretation of the right hand-side of $\mathsf{R}_{\mathbf{r_i}}$ in $\mathcal{M}_{(f_1,\ldots,f_n)}$:

$$\begin{array}{rcl} f'_1 & = & [\![\forall p\ \ldots \Rightarrow \mathbf{rt_1} \Rightarrow \ldots \Rightarrow \epsilon(p)]\!]^{\mathcal{M}_{(f_1,\ldots,f_n)}} \\ & \vdots & \\ f'_n & = & [\![\forall p\ \ldots \Rightarrow \mathbf{rt_n} \Rightarrow \ldots \Rightarrow \epsilon(p)]\!]^{\mathcal{M}_{(f_1,\ldots,f_n)}} \end{array}$$

We extend now the order $\sqsubseteq$ to $\mathcal{E}_\mathcal{M}$ the following way: $\mathcal{M}_{(f_1,\ldots,f_n)} \sqsubseteq \mathcal{M}_{(f'_1,\ldots,f'_n)}$ if $f_1 \sqsubseteq f'_1, \ldots, f_n \sqsubseteq f'_n$. Then by lemma 4, Φ is monotone for $\sqsubseteq$ and thus we can build a fixpoint $\mathcal{M}_{(F_1,\ldots,F_n)}$ of Φ.
We complete this model $\mathcal{M}_{(F_1,\ldots,F_n)}$ by interpreting every $\dot\tau$ by $[\![\tau]\!]$ to obtain $\mathcal{M}$. □

Corollary 3. *For all signature Σ, ML_Σ enjoys the strong normalization property.*

5. Extension to Heyting arithmetic

The next natural extension of this system is to annotate types with dependant information. A dependant version of System T would be Heyting arithmetic. The type of the recursor is then decorated into the following induction scheme, where P is a proposition:

$$P\{0\} \Rightarrow (\forall y\ P\{y\} \Rightarrow P\{\mathsf{S}\ y\}) \Rightarrow \forall n\ P\{n\}$$

Let us see how to encode this system in supernatural deduction modulo as we have done for system T.

We first introduce a unary predicate **N**(n) which expresses the fact that n is a natural number. This is the dependent counterpart of $\mathbf{nat}$. The associated proposition rewrite rule is then:

$$\mathsf{R}_\mathbf{N} : \mathbf{N}(n) \to \forall p\ 0 \in p \Rightarrow \mathit{hered}_\mathbf{N}(p) \Rightarrow n \in p$$

Where hered is defined as follows.

$$\mathrm{hered}_\mathbf{N}(p) = \forall m\ (\mathbf{N}(m) \Rightarrow m \in p \Rightarrow (\mathsf{S}\ m) \in p)$$

The associated super-rules are then

$$(\mathsf{R}_\mathbf{N}I)\ \frac{\Gamma, \alpha : 0 \in x, \beta : \mathrm{hered}_\mathbf{N}(x) \vdash \pi : n \in x}{\Gamma \vdash \lambda \mathsf{R}_\mathbf{N}(x, \alpha, \beta).\pi : \mathbf{N}(n)}\ x \notin \mathcal{FV}(\Gamma)$$

$$(\mathsf{R}_\mathbf{N}E)\ \frac{\Gamma \vdash \pi : \mathbf{N}(n)\ \ \Gamma \vdash \pi_0 : 0 \in t\ \ \Gamma \vdash \pi_s : \mathrm{hered}_\mathbf{N}(t)}{\Gamma \vdash \pi\ \mathsf{R}_\mathbf{N}(t, \pi_0, \pi_s) : n \in t}$$

and the instance of the ρ-calculus reduction rule for $\mathsf{R}_\mathbf{N}$ is still the same as for $\mathsf{R}_\mathbf{nat}$.

$$\lambda \mathsf{R}_\mathbf{N}(x, \alpha, \beta).\pi\ \mathsf{R}_\mathbf{N}(t, \pi_0, \pi_s) \rhd_\rho \pi\{x, \alpha, \beta := t, \pi_0, \pi_s\}$$

The real difference *w.r.t.* the non-dependant version is the type of β which now takes as an argument a first-order term n along with the proof of $\mathbf{N}(n)$ that it is a natural number. This is reflected by the type of 0 and S encodings:

$$\begin{array}{rcl} \nu_0 & = & \lambda \mathsf{R}_\mathbf{N}(x, \alpha, \beta).\alpha : \mathbf{N}(0) \\ \nu_\mathsf{S} & = & \lambda n.\lambda \nu_n.\lambda \mathsf{R}_\mathbf{N}(x, \alpha, \beta).(\beta\ n\ \nu_n\ (\nu_n\ \mathsf{R}_\mathbf{N}(x, \alpha, \beta))) \\ & : & \forall n\ \mathbf{N}(n) \Rightarrow \mathbf{N}(\mathsf{S}\ n) \end{array}$$

An interesting point here is that, contrary to the non-dependent case, the types of ν_0 and ν_S left us no choice

concerning their implementation. Indeed, the only way to get a proof of $\mathbf{N}(0)$ is through α while we are forced to re-use β to inhabit the type of ν_{S}. Since the ρ-reduction rule is generic and not *ad-hoc* for this particular case, this indicates that the computational behaviour of the system is fully contained in the type of the induction scheme, which is a well-known result in System F.

Proving super-consistency of this theory is still possible using the same positivity argument. It has actually be done in [1], which proposes a deduction modulo presentation of Heyting's arithmetic and proves its normalization. The proof is a bit more technical since it requires a more complex description and semantic denotation of the constants encoding the propositions (those meant to instantiate $\forall p$ in $\mathsf{R}_{\mathbf{N}}$). This still can be done in a finite number of rewrite rules using explicit substitutions as described in [20].

An exciting aspect of this later proof is that it not only expresses the induction scheme using a rewrite rule but also equality as well as $+$ and $\times$ definitions, without changing anything to the proof. Put into perspective with the present work, this offers a flexible proof method to prove strong normalization of types systems with recursors mixed with function and proposition symbols defined by rewrite systems, which is a challenging research topic [2, 6].

One last remark concerning equality is that it is usually expressed by a parametrized inductive type which encodes the fact that it is the smallest reflexive predicate in systems like Coq. On the other hand, [1] encodes it by the following rewrite system.

$$\begin{array}{rcl@{\qquad}rcl} 0 = 0 & \to & \top & (\mathsf{S}\ n) = 0 & \to & \bot \\ (\mathsf{S}\ n) = (\mathsf{S}\ m) & \to & n = m & 0 = (\mathsf{S}\ n) & \to & \bot \end{array}$$

We conjecture that every parametrized inductive type can be simulated by a similar proposition rewrite system.

6. Conclusion

We have introduced a new semantic method to prove system T termination by exhibiting an intermediate system between deduction modulo and system T based on supernatural deduction. We then have extended the result to a family of inductive types and have finally shown how the method could be applied to Heyting arithmetic as a dependant version of system T with rewrite rules on proposition.

This is the first step of a generalization of the results presented section 4 to a class of dependant inductive types *à la* Martin-Löf. Another research topic would be the encoding of parametrized inductive types by means of proposition rewrite rules as conjectured above. Finally, it would be interesting to instantiate the proposition rewrite system simulating system T for superdeduction applied to sequent calculus [5]. This probably would provide a good intuition of what a classical calculus with recursors could looks like.

References

[1] L. Allali. Algorithmic equality in Heyting arithmetic modulo. In *TYPES*, 2007. To appear.

[2] F. Blanqui. Definitions by rewriting in the calculus of constructions. In *Logic in Computer Science*, pages 9–18, 2001.

[3] F. Blanqui. Inductive types in the calculus of algebraic constructions, 2003.

[4] P. Brauner, G. Dowek, and B. Wack. Normalization in supernatural deduction and in deduction modulo. `http://www.loria.fr/~brauner/submission_94.pdf`, 2007.

[5] P. Brauner, C. Houtmann, and C. Kirchner. Principles of superdeduction. In *LICS '07: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 41–50, Washington, DC, USA, 2007. IEEE Computer Society.

[6] J. Chrzaszcz and D. Walukiewicz-Chrzaszcz. Towards rewriting in coq. In *Rewriting, Computation and Proof*, pages 113–131, 2007.

[7] H. Cirstea and C. Kirchner. The rewriting calculus — Part I *and* II. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9(3):427–498, May 2001.

[8] H. Cirstea, L. Liquori, and B. Wack. Rewriting calculus with fixpoints: Untyped and first-order systems. In *Proceedings of TYPES*, volume 3085 of *LNCS*. Springer, 2003.

[9] M. DeMarco and J. Lipton. Completeness and cut-elimination in the intuitionistic theory of types. *J. Log. Comput.*, 15(6):821–854, 2005.

[10] G. Dowek. Truth values algebras and proof normalization. In *TYPES*, pages 110–124, 2006.

[11] G. Dowek, T. Hardin, and C. Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31(1):33–72, Nov 2003.

[12] G. Dowek and A. Miquel. Cut elimination for zermelo's set theory. Available on author's web page.

[13] G. Dowek and B. Werner. Proof normalization modulo. *Journal of Symbolic Logic*, 68(4):1289–1316, 2003.

[14] G. Dowek and B. Werner. Arithmetic as a theory modulo. In J. Giesl, editor, *Proceedings of RTA'05*, volume 3467 of *LNCS*, pages 423–437. Springer, 2005.

[15] G. Gentzen. Untersuchungen über das logische schließen. In K. Berka and L. Kreiser, editors, *Logik-Texte: Kommentierte Auswahl zur Geschichte der Modernen Logik (vierte Auflage)*, pages 206–262. Akademie-Verlag, Berlin, 1986.

[16] E. Gimenez. A tutorial on recursive types in coq, 1998.

[17] J.-Y. Girard. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 1989.

[18] O. Hermant. Semantic cut elimination in the intuitionistic sequent calculus. In P. Urzyczyn, editor, *Typed Lambda-Calculi and Applications*, volume 3461 of *Lecture Notes in Computer Science*, pages 221–233, Nara, Japan, 2005. Springer-Verlag.

[19] C. Houtmann. Cohérence de la déduction surnaturelle. Master's thesis, École Normale Supérieure de Cachan, 2006.

[20] F. Kirchner. A finite first-order theory of classes. In *Proc. 2006 Int. Workshop on Proofs and Programs*, Lecture notes in Computer Science. Springer-Verlag, 2006.

[21] U. Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, September 2007.

[22] M. Okada. A uniform semantic proof for cut-elimination and completeness of various first and higher order logics. *Theor. Comput. Sci.*, 281(1-2):471–498, 2002.

[23] M. Parigot. Programming with proofs: A second order type theory. In *ESOP '88: Proceedings of the 2nd European Symposium on Programming*, pages 145–159, London, UK, 1988. Springer-Verlag.

[24] C. Paulin-Mohring. Inductive definitions in the system coq - rules and properties. In *TLCA '93: Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 328–345, London, UK, 1993. Springer-Verlag.

[25] Peter B. Andrews. Resolution in type theory. 36:414–432, 1971.

[26] D. Prawitz. Hauptsatz for higher order logic. *J. Symb. Log.*, 33(3):452–457, 1968.

[27] W. W. Tait. Intensional interpretation of functionals of finite type I. *jsl*, 32:198–212, 1967.

[28] M. Takahashi. A proof of cut-elimination theorem in simple type theory. *Journal of the Mathematical Society of Japan*, 19:399–410, 1967.

[29] B. Werner. *Une Théorie des Constructions Inductives*. PhD thesis, Université Paris 7, 1994.

A. Deduction modulo typing rules

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

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

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

$$(\wedge I)\ \frac{\Gamma \vdash_{\equiv} \pi : A \qquad \Gamma \vdash_{\equiv} \pi' : B}{\Gamma \vdash_{\equiv} \langle \pi, \pi' \rangle : C}\ C \equiv (A \wedge B)$$

$$(\wedge E_2)\ \frac{\Gamma \vdash_{\equiv} \pi : C}{\Gamma \vdash_{\equiv} \mathrm{fst}(\pi) : A}\ C \equiv (A \wedge B)$$

$$(\wedge E_1)\ \frac{\Gamma \vdash_{\equiv} \pi : C}{\Gamma \vdash_{\equiv} \mathrm{snd}(\pi) : B}\ C \equiv (A \wedge B)$$

$$(\vee I_1)\ \frac{\Gamma \vdash_{\equiv} \pi : A}{\Gamma \vdash_{\equiv} i(\pi) : C}\ C \equiv (A \vee B)$$

$$(\vee I_2)\ \frac{\Gamma \vdash_{\equiv} \pi : B}{\Gamma \vdash_{\equiv} i(\pi) : C}\ C \equiv (A \vee B)$$

$$(\vee E)\ \frac{\Gamma \vdash_{\equiv} \pi_1 : D \qquad \Gamma \alpha : A \vdash_{\equiv} \pi_2 : C \qquad \Gamma \beta : B \vdash_{\equiv} \pi_3 : C}{\Gamma \vdash_{\equiv} (\delta\ \pi_1\ \alpha.\pi_2\ \beta.\pi_3) : C}\ D \equiv (A \vee B)$$

$$(\top I)\ \frac{}{\vdash_{\equiv} I : A}\ A \equiv \top$$

$$(\bot E)\ \frac{\Gamma \vdash_{\equiv} \pi : B}{\Gamma \vdash_{\equiv} (\delta_{\bot}\ \pi) : A}\ B \equiv \bot$$

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

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

$$(\exists I)\ \frac{\Gamma \vdash_{\equiv} \pi : A\{x := t\}}{\Gamma \vdash_{\equiv} \langle t, \pi \rangle : B}\ B \equiv (\exists x\ A)$$

$$(\exists E)\ \frac{\Gamma \vdash_{\equiv} \pi : C \qquad \Gamma \alpha : A \vdash_{\equiv} \pi' : B}{\Gamma \vdash_{\equiv} (\delta_{\exists}\ \pi\ x.\alpha.\pi') : B}\ C \equiv (\exists x\ A) \text{ and } x \notin FV(\Gamma, B)$$

Figure 3. Typing rules for deduction modulo with a congruence $\equiv$