A Proof of Strong Normalisation using Domain Theory

Thierry Coquand, Arnaud Spiwack

A proof of strong normalisation using domain theory

Thierry Coquand
Chalmers Tekniska Högskola
Gothenburg
coquand@cs.chalmers.se

Arnaud Spiwack
Ecole Normale Supérieure de Cachan

Arnaud.Spiwack@dptinfo.ens-cachan.fr

Abstract

U. Berger, [11] significantly simplified Tait's normalisation proof for bar recursion [27], see also [9], replacing Tait's introduction of infinite terms by the construction of a domain having the property that a term is strongly normalizing if its semantics is $\neq\bot$. The goal of this paper is to show that, using ideas from the theory of intersection types [2, 6, 7, 21] and Martin-Löf's domain interpretation of type theory [18], we can in turn simplify U. Berger's argument in the construction of such a domain model. We think that our domain model can be used to give modular *proofs of strong normalization for various type theory. As an example, we show in some details how it can be used to prove strong normalization for Martin-Löf dependent type theory extended with bar recursion, and with some form of proof-irrelevance.*

1 Introduction

In 1961, Spector [25] presented an extension of Gödel's system T by a new schema of definition called bar recursion. With this new schema, he was able to give an interpretation of Analysis, extending Gödel's Dialectica interpretation of Arithmetic, and completing preliminary results of Kreisel [17]. Tait proved a normalisation theorem for Spector's bar recursion, by embedding it in a system with infinite terms [27]. In [9], an alternative form of bar recursion was introduced. This allowed to give an interpretation of Analysis by modified realisability, instead of Dialectica interpretation. The paper [9] presented also a normalisation proof for this new schema, but this proof, which used Tait's method of introducing infinite terms, was quite complex. It was simplified significantly by U. Berger [11, 12], who used instead a modification of Plotkin's computational adequacy theorem [22], and could prove *strong* normalisation. In a way, the idea is to replace infinite terms by elements of a domain interpretation. This domain has the property that a term is strongly normalisable if its semantics is $\neq\bot$

The main contribution of this paper is to show that, using ideas from intersection types [2, 6, 7, 21] and Martin-Löf's domain interpretation of type theory [18], one can in turn simplify further U. Berger's argument. Contrary to [11], we build a domain model for an *untyped* programming language. A noteworthy feature of this domain model is that it is in a natural way a *complete* lattice, and in particular it has a *top* element which can be seen as the interpretation of a top-level exception in programming language. We think that this model can be the basis of *modular* proofs of strong normalisation for various type systems. As a main application, we show that Martin-Löf dependent type theory extended with various form of bar recursion has the strong normalisation property. To illustrate further the modularity of this approach, we show the strong normalisation property when adding some form of proof-irrelevance to our type theory [28].

2 An Untyped Programming Language

Our programming language is untyped λ-calculus extended with constants, and has the following syntax.

$$M, N ::= x \mid \lambda x.M \mid M\ N \mid c \mid f$$

There are two kinds of constants: *constructors* $c, c', \ldots$ and *defined constants* $f, g, \ldots$. We use $h, h', \ldots$ to denote a constant which may be a constructor or defined. Each constant has an *arity*, but can be partially applied. We write $\mathsf{FV}(M)$ for the

set of free variables of M. We write $N(x = M)$ the result of substituting the free occurences of x by M in N. and may write it $N[M]$ if x is clear from the context. We consider terms up to α-conversion.

The computation rules of our programming language are the usual β-reduction and *ι-reduction* defined by a set of rewrite rules of the form

$$f\ p_1 \ldots p_k \to M$$

where k is the arity of f and $\mathsf{FV}(M) \subseteq \mathsf{FV}(f\ p_1 \ldots p_k)$. In this rewrite rule, $p_1, \ldots, p_k$ are *constructor patterns i.e.* terms of the form

$$p ::= x \mid c\, p_1 \ldots p_l$$

where l is the arity of c. Like in [11], we assume our system of constant reduction rules to be *left linear*, i.e. a variable occurs at most once in the left hand side of a rule, and *mutually disjoint*, i.e. the left hand sides of two disjoint rules are non-unifiable. We write $M \to M'$ if M reduces in one step to M' by β, ι-reduction and $M =_{\beta,\iota} M'$ if M, M' are convertible by β, ι conversion. It follows from our hypothesis on our system of reduction rules that β, ι-reduction is confluent. We write $\to(M)$ the set of terms M' such that $M \to M'$.

We work with a given set of constants, that are listed in the appendix, but our arguments are general and make use only of the fact that the reduction system is left linear and mutually disjoint. We call UPL, for Untyped Programming Language, the system defined by this list of constants and ι-reduction rules. The goal of the next section is to define a domain model for UPL that has the property that M is strongly normalizing if $[\![M]\!] \neq \bot$.

3 A domain for strong normalization

3.1 Formal Neighbourhoods

Definition 1 *The* Formal Neighbourhoods *are given by the following grammar:*

$$U, V ::= \nabla \mid c\, U_1 \ldots U_k \mid U \to V \mid U \cap V$$

On these neighbourhoods we introduce a *formal inclusion* $\subseteq$ relation defined inductively by the rules of Figure 1. In these rules we use the formal equality relation $U = V$ defined to be $U \subseteq V$ and $V \subseteq U$. We let $\mathcal{M}$ be the set of neighbourhoods quotiented by the formal equality. The terminology "formal neighborhoods" comes from [17, 23, 18].

$$\begin{array}{rcl}
\nabla \cap U & = & \nabla \\
c\, U_1 \ldots U_k \cap c'\, V_1 \ldots V_l & = & \nabla \\
c\, U_1 \ldots U_k \cap V \to W & = & \nabla \\
(U \to V_1) \cap (U \to V_2) & = & U \to (V_1 \cap V_2) \\
c\, U_1 \ldots U_k \cap c\, V_1 \ldots V_k & = & c\, (U_1 \cap V_1) \ldots (U_k \cap V_k)
\end{array}$$

$$\frac{U_1 \subseteq U_2 \quad U_2 \subseteq U_3}{U_1 \subseteq U_3} \qquad \frac{}{U \subseteq U}$$

$$\frac{U \subseteq V_1 \quad U \subseteq V_2}{U \subseteq V_1 \cap V_2} \qquad \frac{}{V_1 \cap V_2 \subseteq V_i}$$

$$\frac{U_2 \subseteq U_1 \quad V_1 \subseteq V_2}{U_1 \to V_1 \subseteq U_2 \to V_2}$$

Figure 1. Formal inclusion

Lemma 1 *The formal inclusion and equality are both* decidable *relations, and $\mathcal{M}$ is a poset for the formal inclusion relation, and $\cap$ defines a binary meet operation on $\mathcal{M}$. We have $c\ U_1 \ldots U_k \neq c'\ V_1 \ldots V_l$ if $c \neq c'$ and $c\, U_1 \ldots U_k = c\, V_1 \ldots V_k$ if and only if $U_1 = V_1, \ldots, U_k = V_k$. An element in $\mathcal{M}$ is either ∇ or of the form $c\, U_1 \ldots U_k$ or of the form $(U_1 \to V_1) \cap \ldots \cap (U_n \to V_n)$ and this defines a partition of $\mathcal{M}$. Furthermore the following "continuity condition" holds: if I finite set and $\bigcap_{i \in I}(U_i \to V_i) \subseteq U \to V$ then the set $J = \{i \in I \mid U \subseteq U_i\}$ is not empty and $\bigcap_{i \in J} V_i \subseteq V$.*

Similar results are proved in [4, 2, 7, 6, 18]. For the proof one can introduce the set of neighborhood in "normal form" by the grammar

$$\begin{array}{lcl}
W, W' & ::= & \nabla \mid c\, W_1\ \ldots\ W_k \mid I \\
I & ::= & (W_1 \to W_1') \cap \cdots \cap (W_n \to W_n')
\end{array}$$

and define directly the operation $\cap$ and the relation $\subseteq$ on this set.

We associate to $\mathcal{M}$ a type system defined in Figure 2 (when unspecified, k is the arity of the related constant). It is a direct extension of the type systems considered in [4, 2, 7, 6, 18]. The typing rules for the constructors and defined constants appear to be new however. Notice that the typing of the function symbols is very close to a recursive definition of the function itself. Also, we make use of the fact that, as a consequence of Lemma 1, one can define when a constructor pattern matches an element of $\mathcal{M}$.

An important consequence of the continuity condition of Lemma 1 is the following result.

$$\frac{x : U \in \Gamma}{\Gamma \vdash_{\mathcal{M}} x : U}$$

$$\frac{}{\Gamma \vdash_{\mathcal{M}} c : U_1 \to \ldots \to U_k \to c\, U_1 \ldots U_k}$$

$$\frac{\Gamma, x{:}U \vdash_{\mathcal{M}} M : V}{\Gamma \vdash_{\mathcal{M}} \lambda x.M : U \to V}$$

$$\frac{\Gamma \vdash_{\mathcal{M}} N : U \to V \quad \Gamma \vdash_{\mathcal{M}} M : U}{\Gamma \vdash_{\mathcal{M}} N\ M : V}$$

$$\frac{\Gamma \vdash_{\mathcal{M}} M : U \quad \Gamma \vdash_{\mathcal{M}} M : V}{\Gamma \vdash_{\mathcal{M}} M : U \cap V}$$

$$\frac{\Gamma \vdash_{\mathcal{M}} M : V \quad V \subseteq U}{\Gamma \vdash_{\mathcal{M}} M : U}$$

$$\frac{\begin{array}{c} f\ p_1 \ldots p_k \to M \quad p_i(W_1, \ldots, W_n) = U_i \\ \Gamma, x_1{:}W_1, \ldots, x_n{:}W_n \vdash_{\mathcal{M}} M : V \end{array}}{\Gamma \vdash_{\mathcal{M}} f : U_1 \to \ldots \to U_k \to V}$$

$$\frac{\begin{array}{c} \text{for any } U_1, \ldots, U_k \text{ such that} \\ \text{no rewriting rules of } f \text{ matches } U_1, \ldots, U_k \end{array}}{\Gamma \vdash_{\mathcal{M}} f : U_1 \to \ldots \to U_k \to \nabla}$$

Figure 2. Types with intersection in $\mathcal{M}$

Lemma 2 *If $\Gamma \vdash_{\mathcal{M}} \lambda x.N : U \to V$ then $\Gamma, x{:}U \vdash_{\mathcal{M}} N : V$.*

3.2 Reducibility candidates

Definition 2 *$\mathcal{S}$ (the set of simple terms) is the set of terms that are neither an abstraction nor a constructor headed term, nor a partially applied destructor headed term (*i.e. *$f\ M_1\ \ldots\ M_n$ is simple if n is greater or equal to the arity of f).*

Definition 3 *A* reducibility candidate *X is a set of terms with the following properties:*

(CR1) $X \subseteq \mathsf{SN}$

(CR2) $\to(M) \subseteq X$ *if* $M \in X$

(CR3) $M \in X$ *if* $M \in \mathcal{S}$ *and* $\to(M) \subseteq X$

It is clear that the reducibility candidates form a complete lattice w.r.t. the inclusion relation. In particular, there is a *least* reducibility candidate R_0, which can be inductively defined as the set of terms $M \in \mathcal{S}$ such that $\to(M) \subseteq R_0$. For instance, if M is a variable x, then we have $M \in R_0$ since $M \in \mathcal{S}$ and $\to(M) = \emptyset$.

We define two operations on sets of terms, which preserve the status of candidates. If c is a constructor of arity k and $X_1, \ldots, X_k$ are sets of terms then the set $c\ X_1 \ldots X_k$ is inductively defined to be the set of terms M of the form $c\, M_1 \ldots M_k$, with $M_1 \in X_1 \ldots M_k \in X_k$ or such that $M \in \mathcal{S}$ and $\to(M) \subseteq c\, X_1 \ldots X_k$. If X and Y are sets of terms, $X \to Y$ is the set of terms N such that $N\ M \in Y$ if $M \in X$.

Lemma 3 *If X and Y are reducibility candidates then so are $X \cap Y$ and $X \to Y$. If $X_1, \ldots, X_k$ are reducibility candidates then so is $c\ X_1 \ldots X_k$.*

Definition 4 *The function $[-]$ associates a reducibility candidate to each formal neighbourhood.*

- $[\nabla] \triangleq R_0$
- $[c\, U_1 \ldots U_k] \triangleq c\, [U_1] \ldots [U_k]$
- $[U \to V] \triangleq [U] \to [V]$
- $[U \cap V] \triangleq [U] \cap [V]$

Lemma 4 *If $U \subseteq V$ for the formal inclusion relation then $[U] \subseteq [V]$ as sets of terms.*

This follows from the fact that all the rules of Figure 1 are valid when we interpret formal neighbourhoods as reducility candidates.

Theorem 5 *If $\vdash_{\mathcal{M}} M : U$ then $M \in [U]$. In particular M is strongly normalising.*

As usual, we prove that if $x_1 : U_1, \ldots, x_n : U_n \vdash_{\mathcal{M}} M : U$ and $M_1 \in [U_1], \ldots, M_n \in [U_n]$ then $M(x_1 = M_1, \ldots, x_n = M_n) \in [U]$. This is direct by induction on derivations using Lemma 4.

3.3 Filter Domain

Definition 5 *An* I-filter[1] *over $\mathcal{M}$ is a subset $\alpha \subseteq \mathcal{M}$ with the following closure properties:*

- *if $U, V \in \alpha$ then $U \cap V \in \alpha$*
- *if $U \in \alpha$ and $U \subseteq V$ then $V \in \alpha$*

It is clear that the set D of all I-filters over $\mathcal{M}$ ordered by the set inclusion is a complete algebraic domain. The finite elements of D are exactly $\emptyset$ and the principal I-filters $\uparrow U \triangleq \{V \mid U \subseteq V\}$. The

[1] This terminology, coming from [6], stresses the fact that the empty set is also an I-filter.

element $\top = \uparrow \nabla$ is the greatest element of D and the least element is $\bot = \emptyset$.

We can define on D a binary application operation

$$\alpha\ \beta \triangleq \{V \mid \exists U, U \to V \in \alpha \wedge U \in \beta\}$$

We have always $\alpha\ \ \bot = \bot$ and $\top\ \beta = \top$ if $\beta \neq \bot$. We write $\alpha_1 \ldots \alpha_n$ for $(\ldots(\alpha_1\ \alpha_2)\ \ldots)\ \alpha_n$.

3.4 Denotational semantics of UPL

As usual, we let $\rho, \nu, \ldots$ to range over *environments*, i.e. mapping from variables to D.

Definition 6 *If M is a term of UPL, $[\![M]\!]_\rho$ is the I-filter of neighborhoods U such that $x_1{:}V_1, \ldots, x_n{:}V_n \vdash_{\mathcal{M}} M : U$ for some $V_i \in \rho(x_i)$ with $\mathsf{FV}(M) = \{x_1, \ldots, x_n\}$.*

A direct consequence of this definition and of Theorem 5 is then

Theorem 6 *If there exists ρ such that $[\![M]\!]_\rho \neq \bot$ then M is strongly normalising.*

Notice also that we have $[\![M]\!]_\rho = [\![M]\!]_\nu$ as soon as $\rho(x) = \nu(x)$ for all $x \in \mathsf{FV}(M)$. Because of this we can write $[\![M]\!]$ for $[\![M]\!]_\rho$ if M is closed. If c is a constructor, we write simply c for $[\![c]\!]$.

Lemma 7 *We have $c\ \alpha_1 \ldots \alpha_k \neq c'\ \beta_1 \ldots \beta_l$ if $c \neq c'$ and $c\ \alpha_1 \ldots \alpha_k = c\ \beta_1 \ldots \beta_k$ if and only if $\alpha_1 = \beta_1 \ldots \alpha_k = \beta_k$, whenever $\alpha_i \neq \bot$, $\beta_j \neq \bot$. An element of D is either $\bot$, or $\top$ or of the form $c\ \alpha_1\ \ldots\ \alpha_k$ with c of arity k and $\alpha_i \neq \bot$ or is a sup of elements of the form $\uparrow(U \to V)$. This defines a partition of D.*

As a consequence of Lemma 7, it is possible to define when a constructor pattern matches an element of D. The next result expresses the fact that we have defined in this way a *strict model* of UPL.

Theorem 8

$$\begin{array}{lcll} [\![x]\!]_\rho & = & \rho(x) & \\ [\![N\ M]\!]_\rho & = & [\![N]\!]_\rho\ [\![M]\!]_\rho & \\ [\![\lambda x.M]\!]_\rho\ \alpha & = & [\![M]\!]_{(\rho, x:=\alpha)} & \text{if } \alpha \neq \bot\ (*) \end{array}$$

If $f\ p_1\ \ldots\ p_k \to M$ and $\alpha_i = [\![p_i]\!]_\rho$ then $[\![f]\!]\ \alpha_1 \ldots \alpha_k = [\![M]\!]_\rho$. If there is no rule for f which matches $\alpha_1, \ldots, \alpha_k$ and $\alpha_1, \ldots, \alpha_k$ are $\neq \bot$ then $[\![f]\!]\ \alpha_1 \ldots \alpha_k = \top$. Finally, if for all $\alpha \neq \bot$ we have $[\![M]\!]_{(\rho, x:=\alpha)} = [\![N]\!]_{(\nu, y:=\alpha)}$ then $[\![\lambda x.M]\!]_\rho = [\![\lambda y.N]\!]_\nu$.

The property $(*)$ follows from Lemma 2.

Corollary 9 $[\![N(x = M)]\!]_\rho = [\![N]\!]_{(\rho, x=[\![M]\!]_\rho)}$

We have for instance $[\![\mathsf{less}]\!]\ (\mathsf{S}\top)\ (\mathsf{S}\top) = \top$, but also $[\![\mathsf{less}]\!]\ \mathsf{nat}\ \mathsf{nat} = \top$. This illustrates the fact that $\top$ can be thought of as the semantics of a top level "error" element.

4 Application to Type Theory

4.1 Typing rules

We follow [19] and present dependent type theory in a Logical Framework extended with some constants. We have three syntactical categories, for *types* $A, B, \ldots$, for *terms* $M, N, \ldots$ and for *contexts* $\Gamma, \Delta, \ldots$ We have a special type Set of (data) types, i.e. primitive types given with constructors. We have also a constructor Fun of arity 2 and we write $(x{:}A) \to B$ instead of $\mathsf{Fun}\ A\ (\lambda x.B)$, and $A \to B$ instead of $\mathsf{Fun}\ A\ (\lambda x.B)$ if x is not free in B. The syntax of the Logical Framework is

$$\begin{array}{lcll} A & ::= & \mathsf{Set} \mid El\ M \mid (x{:}A) \to A & types \\ M & ::= & x \mid M\ M \mid \lambda x.M & terms \\ \Gamma & ::= & () \mid \Gamma, x{:}A & contexts \end{array}$$

The general typing rules of the Logical Framework are presented in figure 3[2]. There are five kinds of judgement $\Delta\ \mathsf{correct}$, $\Delta \vdash A$, $\Delta \vdash M : A$, $\Delta \vdash A_1 = A_2$ and $\Delta \vdash M_1 = M_2 : A$. We write $\Gamma \vdash J$ where J can have the form $A, A = B, M : A, M_1 = M_2 : A$. The constants are the ones of our language UPL, and the typing rules of these constants are also given in the appendix.

The system is designed in such a way that the following lemmas can be directly proved by induction on derivation. For a detailed metatheory of a similar system, see [16].

If γ is a substitution, we write $\gamma : \Delta \to \Gamma$ to express that we have $\Delta \vdash x\gamma : A\gamma$ for all $x{:}A$ in Γ.

Lemma 10 *If $\Delta\ \mathsf{correct}$ and $\gamma : \Delta \to \Gamma$ and $\Gamma \vdash J$ then $\Delta \vdash J\gamma$.*

[2] In this presentation, we consider λ-terms up to α-conversion. This system is quite close to the substitution calculus of P. Martin-Löf [15]. We note however that the following judgement is derivable

$$A{:}\mathsf{Set}, P{:}A \to \mathsf{Set} \vdash \lambda x.\lambda x.x : (x{:}A) \to P\ x \to P\ x$$

while it is not in the substitution calculus (as noticed in [20]).

rules for contexts

$$\frac{}{() \text{ correct}} \qquad \frac{\Gamma \text{ correct} \quad \Gamma \vdash A}{\Gamma, x{:}A \text{ correct}}$$

rules for types

$$\frac{\Gamma \text{ correct}}{\Gamma \vdash \mathsf{Set}} \qquad \frac{\Gamma \vdash M : \mathsf{Set}}{\Gamma \vdash \mathsf{El}\ M} \qquad \frac{\Gamma, x{:}A \vdash B}{\Gamma \vdash (x{:}A) \to B}$$

rules for terms

$$\frac{\Gamma \text{ correct} \quad (x{:}A) \in \Gamma}{\Gamma \vdash x{:}A} \qquad \frac{\Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x.M : (x{:}A) \to B} \qquad \frac{\Gamma \vdash N : (x{:}A) \to B \quad \Gamma \vdash M : A}{\Gamma \vdash N\ M : B[M]}$$

type equality rule

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A = B}{\Gamma \vdash M : B}$$

substitution rule

$$\frac{\Gamma, x{:}A \vdash B \quad \Gamma \vdash M_1 = M_2 : A}{\Gamma \vdash B[M_1] = B[M_2]}$$

conversion rules

$$\frac{\Gamma \vdash A}{\Gamma \vdash A = A} \qquad \frac{\Gamma \vdash A = B}{\Gamma \vdash B = A} \qquad \frac{\Gamma \vdash A = B \quad \Gamma \vdash B = C}{\Gamma \vdash A = C}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash M = M : A} \qquad \frac{\Gamma \vdash M = N : A}{\Gamma \vdash N = M : A} \qquad \frac{\Gamma \vdash M = N : A \quad \Gamma \vdash N = P : A}{\Gamma \vdash M = P : A}$$

$$\frac{\Gamma \vdash M = N : A \quad \Gamma \vdash A = B}{\Gamma \vdash M = N : B}$$

$$\frac{\Gamma \text{ correct}}{\Gamma \vdash \mathsf{Set} = \mathsf{Set}} \qquad \frac{\Gamma \vdash M_1 = M_2 : \mathsf{Set}}{\Gamma \vdash \mathsf{El}\ M_1 = \mathsf{El}\ M_2} \qquad \frac{\Gamma \vdash A_1 = A_2 \quad \Gamma, x{:}A_1 \vdash B_1 = B_2}{\Gamma \vdash (x{:}A_1) \to B_1 = (x{:}A_2) \to B_2}$$

$$\frac{\Gamma, x{:}A \vdash M_1 = M_2 : B}{\Gamma \vdash \lambda x.M_1 = \lambda x.M_2 : (x{:}A) \to B}$$

$$\frac{\Gamma, x{:}A \vdash B \quad \Gamma \vdash N_1 = N_2 : (x{:}A) \to B \quad \Gamma \vdash M_1 = M_2 : A}{\Gamma \vdash N_1\ M_1 = N_2\ M_2 : B[M_1]}$$

$$\frac{\Gamma, x{:}A \vdash N : B \quad \Gamma \vdash M : A}{\Gamma \vdash (\lambda x.N)\ M = N[M] : B[M]} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash M : (x{:}A) \to B}{\Gamma \vdash M = \lambda x.M\ x : (x{:}A) \to B}$$

$$\frac{\Gamma \vdash M : A \quad M \to_\iota N \quad \text{(toplevel reduction)}}{\Gamma \vdash M = N : A}$$

Figure 3. Logical Framework

Lemma 11 *If* $\Gamma \vdash (x{:}A_1) \to B_1 = (x{:}A_2) \to B_2$ *then* $\Gamma \vdash A_1 = A_2$ *and* $\Gamma, x{:}A_1 \vdash B_1 = B_2$.

Lemma 12 *If* $\Gamma \vdash A = B$ *then* $\Gamma \vdash A$ *and* $\Gamma \vdash B$. *If* $\Gamma \vdash M_1 = M_2 : A$ *then* $\Gamma \vdash M_1 : A$ *and* $\Gamma \vdash M_2 : A$. *If* $\Gamma \vdash M : A$ *then* $\Gamma \vdash A$. *If* $\Gamma \vdash M = N : A$ *then* $\Gamma \vdash A$.

Corollary 13 *If* $\Gamma \vdash M : A$ *and* $M \to M'$ *then* $\Gamma \vdash M' : A$

Corollary 14 *If* $\Gamma \vdash M : A$ *and* $\Gamma \vdash M' : A$ *and* $M =_{\beta,\iota} M'$ *then* $\Gamma \vdash M = M' : A$

This is direct from Corollary 13 and the Church-Rosser property of β, ι reduction.

Notice that, because of the conversion rule, the strengthening property, stating that $\Gamma \vdash J$ follows from $\Gamma, x{:}A \vdash J$ if x is not free in J, is not clear *a priori*. It is actually a consequence of the normalisation property.

4.2 PER Models of type theory

A *partial equivalence relation* on D is a subset $X \subseteq \mathsf{D}$ with an equivalence relation $=_X$ on X. We write $u_1 = u_2 \in X$ instead of $u_1 =_X u_2$. We let $\mathsf{PER}(\mathsf{D})$ be the collection of all partial equivalence relation on D. If F is a function from X to $\mathsf{PER}(\mathsf{D})$ such that $F(u_1) = F(u_2)$ whenever $u_1 = u_2 \in X$ we write $F : X \to \mathsf{PER}(\mathsf{D})$. If $X \in \mathsf{PER}(\mathsf{D})$ and $F : X \to \mathsf{PER}(\mathsf{D})$ we define $\Pi(X, F) \in \mathsf{PER}(\mathsf{D})$ by

$v \in \Pi(X, F)$ if and only if $u_1 = u_2 \in X$ implies $v\ u_1 = v\ u_2 \in F(u_1)$

$v_1{=}v_2 \in \Pi(X, F)$ if and only if $v_1\ u{=}\ v_2\ u \in F(u)$ for all $u \in X$

These constructions are standard [5].

Definition 7 *A* PER model *of our type theory consists of a pair* T, I *with* $T \in \mathsf{PER}(\mathsf{D})$ *and* $I : T \to \mathsf{PER}(\mathsf{D})$ *is such that*

1. $\mathsf{Set} \in T$
2. *if* $U_1 = U_2 \in T$ *and* $u_1 = u_2 \in I(U)$ *implies* $F_1\ u_1 = F_2\ u_2 \in T$ *then* $\mathsf{Fun}\ U_1\ F_1 = \mathsf{Fun}\ U_2\ F_2 \in T$ *and* $I(\mathsf{Fun}\ U_1\ F_1) = I(\mathsf{Fun}\ U_2\ F_2) = \Pi(I(U_1), \lambda u.I(F_1\ u))$
3. $\mathsf{El}\ u_1 = \mathsf{El}\ u_2 \in T$ *if* $u_1 = u_2 \in I(\mathsf{Set})$.
4. $[\![A]\!] \in T$ *and* $h \in I([\![A]\!])$ *whenever* $\vdash h{:}A$ *is a typing rule for the constant* h.

If Δ is a context we write $\rho \Vdash \Delta$ to express that $[\![A]\!]_\rho \in T$ and $\rho(x) \in I([\![A]\!]_\rho)$ for $x{:}A$ in Δ and we write $\rho_1 = \rho_2 \Vdash \Delta$ to express that $[\![A]\!]_{\rho_1} = [\![A]\!]_{\rho_2} \in T$ and $\rho_1(x) = \rho_2(x) \in [\![A]\!]_{\rho_1}$ for $x{:}A$ in Δ.

The next result states the soundness of PER semantics for the type system. We assume given a PER model T, I of our type theory. The proof is direct by induction on derivations using Theorem 8 and Lemma 9.

Theorem 15 *Assume* $\rho_1 = \rho_2 \Vdash \Delta$. *If* $\Delta \vdash A$ *then* $[\![A]\!]_{\rho_1} = [\![A]\!]_{\rho_2} \in T$. *If* $\Delta \vdash M{:}A$ *then* $[\![A]\!]_{\rho_1} = [\![A]\!]_{\rho_2} \in T$ *and* $[\![M]\!]_{\rho_1} = [\![M]\!]_{\rho_2} \in I([\![A]\!]_{\rho_1})$.

A *totality relation* on D is a partial equivalence relation X such that $u \not= \bot$ if $u \in X$ and $\top \in X$. We let $\mathsf{TR}(\mathsf{D})$ be the collection of all totality relations.

Lemma 16 *If* $X \in \mathsf{TR}(\mathsf{D})$ *and* $F : X \to \mathsf{TR}(\mathsf{D})$ *then* $\Pi(X, F) \in \mathsf{TR}(\mathsf{D})$.

We have $\top \in X$. If $v \in \Pi(X, F)$ then $v\ \top \in F(\top)$ and so $v\ \top \not= \bot$ and $v \not= \bot$ holds. If $u \in X$ then $u \not= \bot$ so that $\top\ u = \top \in F(u)$. This shows $\top \in \Pi(X, F)$.

The next theorem has a subtle proof, but it is standard [1, 8, 24]. The main idea is to define the pair T, I by an inductive process, using Lemma 7 to ensure the consistency of this definition.

Theorem 17 *The filter model* D *of UPL can be extended to a model of our type theory, in such a way that* $T \in \mathsf{TR}(\mathsf{D})$ *and* $I : T \to \mathsf{TR}(\mathsf{D})$.

For instance the element nat is in T and $I(\mathsf{nat})$ is the PER containing the elements $\mathsf{S}^k\ \mathsf{0}$ and $\mathsf{S}^k\ \top$. Similarly, Void will be in T and $I(\mathsf{Void})$ contains only $\top$.

The verifications of condition 4 of Definition 7 for the constants Φ and Ψ are similar to the ones in [11], and it is crucial at this point that we are using a domain model. These constants make also the system proof-theoretically strong, at least the strength of second-order arithmetic.

Corollary 18 *If* $\vdash A$ *then* $[\![A]\!] \not= \bot$. *If* $\vdash M : A$ *then* $[\![M]\!] \not= \bot$.

By combining Corollary 18 with Theorem 6 we get

Theorem 19 *If* $\vdash A$ *then* A *is strongly normalisable. If* $\vdash M : A$ *then* M *is strongly normalisable.*

4.3 Decidability properties

In order to get decidability of conversion, we use a technique introduced in [14] and first define the η-expansion $\eta\ A\ M$ in a syntactical way.

$$\eta\ \mathsf{Set}\ M = M \qquad \eta\ (\mathsf{El}\ B)\ M = M$$
$$\eta\ (\mathsf{Fun}\ A\ F)\ M =$$
$$\lambda x.\eta\ (F\ (\eta\ A\ x))\ (M\ (\eta\ A\ x))$$

Lemma 20 *If* $\Gamma \vdash M : A$ *then* $\Gamma \vdash M = \eta\ A\ M : A$

The intuition behind the next statement is clear: if we work only with the η-expansions of the terms, we don't need the η-conversion rule. For a precise proof, we rely on the soundness of a particular PER model for our type system.

Lemma 21 *If* $\vdash M_1 = M_2 : A$ *then we have* $\eta\ A\ M_1 =_{\beta,\iota} \eta\ A\ M_2$

For the proof we use the following PER model. The domain D is the set of all terms, with β, ι-conversion as equality. The PER Set is interpreted by the conversion: we have $M_1 = M_2 : \mathsf{Set}$ if and only if $M_1 = M_2$, and for any M the PER $\mathsf{El}\ M$ is also the conversion. A constant h defined to be of type A is interpreted by $\eta\ A\ h$, and one can check that $\eta\ A\ f$ satisfies the same equality as f. For instance, Rec is interpreted as $\lambda C.\lambda a.\lambda b.\lambda n.\mathsf{Rec}\ (\lambda x.C\ x)\ a\ (\lambda x.\lambda y.b\ x\ y)\ n$. The soundness of the type theory w.r.t. this interpretation gives the result.

Theorem 22 *If* $\vdash M_1 : A$ *and* $\vdash M_2 : A$ *then* $\vdash M_1 = M_2 : A$ *if and only if* $\eta\ A\ M_1 =_{\beta,\iota} \eta\ A\ M_2$.

This follows from Lemmas 20 and 21 and Corollary 14.

Corollary 23 *If* $\vdash M_1 : A$ *and* $\vdash M_2 : A$ *then* $\vdash M_1 = M_2 : A$ *is decidable.*

Indeed, by the theorem we are reduced to check $\eta\ A\ M_1 =_{\beta,\iota} \eta\ A\ M_2$. This is decidable since both $\eta\ A\ M_1$ and $\eta\ A\ M_2$ are strongly normalisable, and β, ι reduction is confluent.

Corollary 24 *If* A *is in* β*-normal form then* $\vdash A$ *is decidable. If* $\vdash A$ *and* M *is in* β*-normal form then* $\vdash M : A$ *is decidable.*

4.4 Proof irrelevance

We add two new constants Prf and O with the rules

$$\frac{\Gamma \vdash M : \mathsf{Set}}{\Gamma \vdash \mathsf{Prf}\ M} \qquad \frac{\Gamma \vdash M : \mathsf{Prf}\ N}{\Gamma \vdash \mathsf{O} : \mathsf{Prf}\ N}$$

$$\frac{\Gamma \vdash M_1 = M_2 : \mathsf{Set}}{\Gamma \vdash \mathsf{Prf}\ M_1 = \mathsf{Prf}\ M_2}$$

$$\frac{\Gamma \vdash M_1 : \mathsf{Prf}\ N \qquad \Gamma \vdash M_2 : \mathsf{Prf}\ N}{\Gamma \vdash M_1 = M_2 : \mathsf{Prf}\ N}$$

We can read the judgement $\mathsf{O} : \mathsf{Prf}\ A$ as claiming that the proposition A is *true*: we know that A has a proof but the proof has been hidden.

Notice that the strengthening property does not hold for this system. We shall be interested however only in terms that do not contain O. This element is only here in order to prove the decidability of the conversion relation and it is not needed in order to have a strongly normalising proof-irrelevant theory.

The PER model extends directly to this system by interpreting O by $\top$ and letting $I([\![\mathsf{Prf}\ A]\!])$ be the set $I([\![\mathsf{El}\ A]\!])$ with the universal equivalence relation. One can show the soundness of this PER model w.r.t. the typing rules and it follows that strong normalisation still holds for this system.

This PER model validates also the following rule.

$$\frac{\Gamma \vdash M : \mathsf{El}\ A}{\Gamma \vdash M : \mathsf{Prf}\ A}$$

For proving the decidability of convertibility, we update the definition of $\eta\ A\ M$ by taking $\eta\ (\mathsf{Prf}\ B)\ M = \mathsf{O}$. It is then still the case that $\vdash M_1 = M_2 : A$ if and only if $\eta\ A\ M_1 =_{\beta,\iota} \eta\ A\ M_2$ if $\vdash M_1 : A$ and $\vdash M_2 : A$.

Theorem 25 *If* A *is in* β*-normal form and does not contain* O *then* $\vdash A$ *is decidable. If* $\vdash A$ *and* M *is in* β*-normal form and does not contain* O *then* $\vdash M : A$ *is decidable.*

5 Conclusion

We have built a filter model D for an untyped calculus having the property that a term is strongly normalisable whenever its semantics is $\neq \bot$, and then used this to give various *modular* proofs of strong normalization. While each part uses essentially variation on standard materials, our use of filter models seems to be new and can be seen as an application of computing science to proof theory. It is interesting that we are naturally lead in

this way to consider a domain with a top element. We have shown on some examples that this can be used to prove strong normalisation theorem in a modular way, essentially by reducing this problem to show the soundness of a PER semantics over the domain D. As suggested to us by Andreas Abel, it seems likely that Theorem 5 has a purely combinatorial proof, similar in complexity to the one for simply typed λ-calculus. There should be no problem to use our model to give a simple normalisation proof of system F extended with bar recursion. For this, we don't need to work with PERs, but it would be enough to work with *totality predicates* that are subsets $X \subseteq \mathsf{D}$ such that $\top \in X$ and $u \neq \bot$ if $u \in X$. It is then direct that totality predicates are closed under arbitrary non empty intersections. By working in the D-set model instead of the PER model over D [26, 3], one should be able to get also strong normalisation theorems for various impredicative type theories extended with bar recursion.

For proving normalisation for *predicative* type systems, the use of the model D is proof-theoretically too strong: the PER are relations over filters, that are themselves sets of formal neighborhoods, and so are essentially third-order objects. For applications not involving strong schemas like bar recursion, it is possible however to work instead only with the definable elements of the set D, and PER becomes second-order objects, as usual. It is then natural to extend our programming language with an extra element $\top$ that plays the role of a top-level error.

A natural extension of this work would be also to state and prove a *density* theorem for our denotational semantics, following [13]. The first step would be to define when a formal neighborhood is of a given type.

In [6, 21], for untyped λ-calculus without constants, it is proved that a term M is strongly normalizing if *and only if* $[\![M]\!] \neq \bot$. This does not hold here since we have for instance $\mathsf{0}\ \mathsf{nat}$ strongly normalizing, but $[\![\mathsf{0}\ \mathsf{nat}]\!] = \bot$. However, it may be possible to find a natural subset of term M for which the equivalence between M is strongly normalizing and $[\![M]\!] \neq \bot$ holds.

A more natural extension of a system with dependent types with some form of bar recursion would be to add a constant for the double negation shift [25], which would be a constant of type

$$((n : \mathsf{nat}) \to \neg(\neg F\ n)) \to \neg\neg((n : \mathsf{nat}) \to F\ n)$$

with suitable computation rules, for $F : \mathsf{nat} \to \mathsf{Set}$. We leave this for further work.

Most of our results hold without the hypotheses that the rewriting rules are mutually disjoint. We only have to change the typing rules for a constant f in Figure 2 by the uniform rule: $\Gamma \vdash_{\mathcal{M}} f : U_1 \to \ldots \to U_k \to V$ if *for all* rules $f\ p_1 \ldots p_k \to M$ and *for all* $W_1, \ldots, W_n$ such that $p_i(W_1, \ldots, W_n) = U_i$ we have $\Gamma, x_1 : W_1, \ldots, x_n : W_n \vdash_{\mathcal{M}} M : V$. (This holds for instance trivially in the special case where no rules for f matches $U_1, \ldots, U_n$.) For instance, we can add a constant $+$ with rewrite rules

$$\begin{array}{llllll} + & n & \mathsf{0} & \to & n \\ + & \mathsf{0} & n & \to & n \\ + & n & (\mathsf{S}\ m) & \to & \mathsf{S}\ (+\ n\ m) \\ + & (\mathsf{S}\ n) & m & \to & \mathsf{S}\ (+\ n\ m) \end{array}$$

and Theorem 6 is still valid with this extension.

Appendix: the language UPL

The constructors $\mathsf{Set}, \mathsf{nat}, \mathsf{Void}, \mathsf{Unit}, \mathsf{0}, []$ (arity 0), $\mathsf{list}, \mathsf{El}, \mathsf{S}$ (arity 1) and $\mathsf{cons}, \mathsf{Fun}$ (arity 2).

The defined constants of the language UPL are $\mathsf{isZero}, \mathsf{less}, \mathsf{length}, \mathsf{concat}, \mathsf{get}, \mathsf{Rec}, \Phi, \Psi$. The arities are clear from the following ι-rules.

$$\begin{array}{l} \mathsf{Rec}\ M\ N\ \mathsf{0} \to N \\ \mathsf{Rec}\ M\ N\ (\mathsf{S}\ P) \to M\ P\ (\mathsf{Rec}\ M\ N\ P) \\ \mathsf{isZero}\ \mathsf{0} \to \mathsf{Unit} \\ \mathsf{isZero}\ (\mathsf{S}\ P) \to \mathsf{Void} \\ \mathsf{less}\ P\ \mathsf{0} \to \mathsf{0} \\ \mathsf{less}\ \mathsf{0}\ (\mathsf{S}\ P) \to \mathsf{S}\ \mathsf{0} \\ \mathsf{less}\ (\mathsf{S}\ P)\ (\mathsf{S}\ Q) \to \mathsf{less}\ P\ Q \\ \mathsf{get}\ a\ (\mathsf{cons}\ x\ L)\ (\mathsf{S}\ N) \to \mathsf{get}\ L\ N \\ \mathsf{get}\ a\ (\mathsf{cons}\ x\ L)\ (\mathsf{0}) \to x \\ \mathsf{get}\ a\ []\ N \to a \\ \mathsf{length}\ [] \to \mathsf{0} \\ \mathsf{length}\ (\mathsf{cons}\ P\ L) \to \mathsf{S}\ (\mathsf{length}\ L) \\ \mathsf{concat}\ []\ x \to \mathsf{cons}\ x\ [] \\ \mathsf{concat}\ (\mathsf{cons}\ P\ L)\ x \to \mathsf{cons}\ P\ (\mathsf{concat}\ L\ x) \\ \Phi\ M\ N\ L\ R \to \\ M\ (\lambda x.\Psi\ M\ N\ L\ R\ x\ (\mathsf{less}\ x\ (\mathsf{length} L))) \\ \Psi\ M\ N\ L\ R\ Q\ \mathsf{0} \to \mathsf{get}\ R\ L\ Q \\ \Psi\ M\ N\ L\ R\ Q\ (\mathsf{S}\ \_) \to \\ N\ Q\ (\lambda x.\Phi\ M\ N\ (\mathsf{concat}\ x\ L))\ R \end{array}$$

Finally the typing rules for constants are the following

$$
\begin{array}{ll}
\mathsf{nat}: & \mathsf{Set} \\
0: & \mathsf{nat} \\
\mathsf{S}: & \mathsf{nat} \to \mathsf{nat} \\
\mathsf{Rec}: & ((n:\mathsf{nat}) \to C\ n \to (C\ (\mathsf{S}\ n))) \to \\
 & C\ 0 \to (n:\mathsf{nat}) \to C\ n \\
 & [C:\mathsf{nat} \to \mathsf{Set}] \\
\mathsf{isZero}: & \mathsf{nat} \to \mathsf{Set} \\
\mathsf{list}: & \mathsf{Set} \to \mathsf{Set} \\
[]: & \mathsf{list}\ A\ [A:\mathsf{Set}] \\
\mathsf{cons}: & A \to \mathsf{list}\ A \to \mathsf{list}\ A\ [A:\mathsf{Set}] \\
\mathsf{Void}: & \mathsf{Set} \\
\mathsf{Unit}: & \mathsf{Set} \\
0: & \mathsf{Unit} \\
\mathsf{get}: & A \to \mathsf{list}\ A \to \mathsf{nat} \to A\ [A:\mathsf{Set}] \\
\mathsf{length}: & \mathsf{list}\ A \to \mathsf{nat}\ [A:\mathsf{Set}] \\
\mathsf{less}: & \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}\ [A:\mathsf{Set}] \\
\mathsf{concat}: & \mathsf{list}\ A \to A \to \mathsf{list}\ A\ [A:\mathsf{Set}] \\
\Phi: & ((\mathsf{nat} \to A) \to \mathsf{nat}) \to \\
 & (\mathsf{nat} \to (A \to \mathsf{nat}) \to A) \to \\
 & \mathsf{list}\ A \to A \to \mathsf{nat}\ [A:\mathsf{Set}] \\
\Psi: & ((\mathsf{nat} \to A) \to \mathsf{nat}) \to \\
 & (\mathsf{nat} \to (A \to \mathsf{nat}) \to A) \to \\
 & \mathsf{list}\ A \to A \to \\
 & \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}\ [A:\mathsf{Set}]
\end{array}
$$

The rules for Φ and Ψ are the rules for a variation of bar recursion. We do not comment further on these rules, since they are essentially the same as the ones given in [11], with the get function suitably modified to take into account the empty type, which is not present in the system [11].

The rule $\mathsf{cons}: A \to \mathsf{list}\ A \to \mathsf{list}\ A\ [A:\mathsf{Set}]$ for instance means that we have

$$\frac{\vdash A:\mathsf{Set}}{\vdash \mathsf{cons}: A \to \mathsf{list}\ A \to \mathsf{list}}$$

and similarly for the other constants.

Acknowledgement

Thanks to Mariangiola Dezani-Ciancaglini for the reference to the paper [6].

References

[1] P. Aczel Frege structures and the notions of proposition, truth and set. *The Kleene Symposium*, pp. 31–59, Stud. Logic Foundations Math., 101, North-Holland, Amsterdam-New York, 1980.

[2] Y. Akama. SN Combinators and Partial Combinatory Algebras. LNCS 1379, p. 302-317, 1998.

[3] Th. Altenkirch. *Constructions, Inductive Types and Strong Normalization.* PhD thesis, University of Edinburgh, 1993.

[4] R. Amadio and P.L. Curien. *Domains and Lambda-Calculi.* Cambridge tracts in theoretical computer science, 46, (1997).

[5] D. Aspinall. Subtyping Dependent Types. CSL'94, LNCS 933, 1994.

[6] S. van Bakel. Complete restrictions of the Intersection Type Discipline. Theoretical Computer Science, 102:135-163, 1992.

[7] H. Barendregt, M. Coppo and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *J. Symbolic Logic* 48 (1983), no. 4, 931–940 (1984).

[8] M. Beeson. *Foundations of constructive mathematics. Metamathematical studies.* Ergebnisse der Mathematik und ihrer Grenzgebiete (3) [Results in Mathematics and Related Areas (3)], 6. Springer-Verlag, Berlin, 1985.

[9] S. Berardi, M. Bezem and Th. Coquand. On the computational content of the axiom of choice. Journal of Symbolic Logic 63 (2), 600-622, 1998.

[10] U. Berger and P. Oliva. Modified Bar Recursion and Classical Dependent Choice. Logic Colloquium '01, 89–107, Lect. Notes Log., 20, Assoc. Symbol. Logic, Urbana, IL, 2005.

[11] U. Berger. Continuous Semantics for Strong Normalisation. LNCS 3526, 23-34, 2005.

[12] U. Berger. A Computational Interpretation of Open Induction. Proceeding of LICS 2004.

[13] U. Berger. Continuous Functionals of Dependent and Transfinite Types. in *Models and Computability*, London Mathematical Society, Lecture Note Series, p. 1–22, 1999.

[14] Th. Coquand, R. Pollack and M. Takeyama. A Logical Framework with Dependently Typed Record. Fundam. Inform. 65 (1-2), p. 113-134, 2005.

[15] D. Fridlender. A proof-irrelevant model of Martin-Lf's logical framework. Math. Structures Comput. Sci. 12 (2002), no. 6, 771–795.

[16] R Harper, F Pfenning. On equivalence and canonical forms in the LF type theory ACM Transactions on Computational Logic, p. 61–101, 2005.

[17] G. Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In *Constructivity in Mathematics*, North-Holland, 1958.

[18] P. Martin-Löf. Lecture note on the domain interpretation of type theory. *Workshop on Semantics of Programming Languages, Chalmers*, (1983).

[19] B Nordstrom, K Petersson, JM Smith. Martin-Löf Type Theory. Handbook of Logic in Computer Science, 2000.

[20] R. Pollack. Closure Under Alpha-Conversion. TYPES 1993, Lecture Notes in Computer Science 806, p. 313-332, 1993.

[21] G. Pottinger. A type assignment for the strongly normalizable terms. in: J.P. Seldin and J.R. Hindley (eds.), *To H. B. Curry: essays on combinatory logic, lambda calculus and formalism*, Academic Press, London, pp. 561-577, 1980.

[22] G. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223-255, 1977.

[23] D. Scott. Lectures on a mathematical theory of computation. Theoretical foundations of programming methodology (Munich, 1981), 145–292, NATO Adv. Study Inst. Ser. C: Math. Phys. Sci., 91, Reidel, Dordrecht, 1982.

[24] D. Scott. Combinators and classes. *λ-calculus and computer science theory*, pp. 1–26. Lecture Notes in Comput. Sci., Vol. 37, Springer, Berlin, 1975.

[25] C. Spector. Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles in current intuitionistic mathematics. In F.D.E.Dekker, editor, Recursive Function Theory, 1962

[26] Th. Streicher. *Semantics of Type Theory.* in the series Progress in Theoretical Computer Science. Basel: Birkhaeuser. XII, 1991.

[27] W.W. Tait. Normal form theorem for bar recursive functions of finite type. *Proceedings of the Second Scandinavian Logic Symposium*, North-Holland, 1971.

[28] B. Werner. A Proof-Irrelevant Type Theory. Unpublished manuscript, 2003.