Tait in one big step

Thorsten Altenkirch and James Chapman
School of Computer Science, University of Nottingham
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
{*txa,jmc*}*@cs.nott.ac.uk*

Abstract

We present a Tait-style proof to show that a simple functional normaliser for a combinatory version of System T terminates. Using a technique pioneered by Bove and Capretta, we can implement the normaliser in total Type Theory. The main interest in our construction is methodological, it is an alternative to the usual small-step operational semantics on the one side and normalisation by evaluation on the other. The present work is motivated by our longer term goal to verify implementations of Type Theory such as Epigram.

Keywords: Normalisation,Strong Computability

1. INTRODUCTION

Traditionally, decidability of equality for typed λ-calculi is established by showing strong normalisation for a small-step reduction relation [21, 15]. However, this is not the only way to establish this result. To show decidability it is sufficient to construct a normalisation function which, for any term, calculates a normal form that is unique for the term's equivalence class. Indeed, normalisation by evaluation (NBE) [8, 2, 3, 1, 6] uses a constructive denotational semantics to construct a normalisation function by inverting the evaluation functional. While NBE has a number of advantages – it is elegant and theoretically well understood – it is hard to tinker with the normalisation function and the decision procedure, e.g. if we want to exploit symbolic equalities to avoid having to normalise terms altogether.

In the present paper we investigate yet another alternative: we directly implement a partial normalisation function and use a Tait-style construction to establish that the function terminates for all typable terms. We use a very simple calculus to demonstrate the approach: a combinatory version of System T. Our goal here is not to establish a new result but to show how Tait's proof can be adapted to show normalisation of big-step reduction and how to implement the normaliser in Type Theory using the technique of Bove and Capretta [9]. Using Tait's method to prove normalisation isn't new either, [16] uses a Tait style proof to show normalisation for a more sophisticated system. The restriction to a combinatory calculus is appropriate since more extensional equalities like full $\beta\eta$ can be decided using a combination of a structural equivalence and weak normalisation, using a variant of the big-step semantics presented here as one component.

This work here is motivated by our goal to verify important aspects of the implementation of Epigram [19, 17, 5], e.g. to show that Epigram's Type Theory ETT (when suitably stratified) [11] is decidable. While the current implementation of ETT uses NBE to implement weak reduction, it is likely that we will move to a big-step reduction similar to the one presented here to be able to optimize the equality test more easily. At the same time, we are using Epigram[18, 5] as a metalanguage to formalise the development presented here.

2. COMBINATORY SYSTEM T

We start by introducing our calculus, using Epigram as our metalanguage. Its types are given by a simple inductive definition using natural deduction style declarations of constants:

$$\underline{\mathsf{data}}\quad \frac{}{\mathsf{Ty}\ :\ \star}\quad \underline{\mathsf{where}}\quad \frac{}{\mathsf{N}\ :\ \mathsf{Ty}}\quad \frac{\sigma\ :\ \mathsf{Ty}\quad \tau\ :\ \mathsf{Ty}}{\sigma{\to}\tau\ :\ \mathsf{Ty}}$$

We will overload the symbol $\to$ to refer to both the object function arrow that we define here and also the metatheoretic function arrow. Which one mean will be clear from the context. The symbol $\star$ is Epigram's type of types and we declare all new symbols as Epigram data or let definitions. Terms are defined as an inductive family indexed by types;

$$\underline{\mathsf{data}}\quad \frac{\sigma\ :\ \mathsf{Ty}}{\mathsf{Tm}_\sigma\ :\ \star}\quad \underline{\mathsf{where}}\quad \frac{}{\mathsf{K}\ :\ \mathsf{Tm}_{\sigma\to\tau\to\sigma}}\quad \frac{}{\mathsf{S}\ :\ \mathsf{Tm}_{(\sigma\to\tau\to\rho)\to(\sigma\to\tau)\to\sigma\to\rho}}$$

$$\frac{t\ :\ \mathsf{Tm}_{\sigma\to\tau}\quad u\ :\ \mathsf{Tm}_\sigma}{t\,u\ :\ \mathsf{Tm}_\tau}\quad \frac{}{0\ :\ \mathsf{Tm}_\mathsf{N}}\quad \frac{}{\mathsf{suc}\ :\ \mathsf{Tm}_{\mathsf{N}\to\mathsf{N}}}\quad \frac{}{\mathsf{prec}\ :\ \mathsf{Tm}_{(\mathsf{N}\to\sigma\to\sigma)\to\sigma\to\mathsf{N}\to\sigma}}$$

The conversion relation is inductively defined, we present it here as an inductively defined family of types — however, this family should be regarded as propositional since we are not interested in the choice of derivations.

$$\underline{\mathsf{data}}\quad \frac{t,u\ :\ \mathsf{Tm}_\sigma}{t\ \simeq\ u\ :\ \mathsf{Prop}}\quad \underline{\mathsf{where}}\quad \frac{}{\mathsf{cK}\ :\ \mathsf{K}\,x\,y\ \simeq\ x}\quad \frac{}{\mathsf{cS}\ :\ \mathsf{S}\,x\,y\,z\ \simeq\ x\,z\,(y\,z)}$$

$$\frac{}{\mathsf{cprec0}\ :\ \mathsf{prec}\,f\,z\,0\ \simeq\ z}\quad \frac{}{\mathsf{cprecsuc}\ :\ \mathsf{prec}\,f\,z\,(\mathsf{suc}\,n)\ \simeq\ f\,n\,(\mathsf{prec}\,f\,z\,n)}\quad \frac{}{\mathsf{crefl}\ :\ t\ \simeq\ t}$$

$$\frac{p\ :\ t\ \simeq\ u}{\mathsf{csym}\,p\ :\ u\ \simeq\ t}\quad \frac{p\ :\ t\ \simeq\ u\quad q\ :\ u\ \simeq\ v}{\mathsf{ctrans}\,p\,q\ :\ t\ \simeq\ v}\quad \frac{p\ :\ t\ \simeq\ v\quad q\ :\ u\ \simeq\ w}{\mathsf{ccong}\,p\,q\ :\ t\,u\ \simeq\ v\,w}$$

Here we use Prop just as a different name for $\star$. However, the intention is that inhabitants of Prop are proof-irrelevant, i.e. have at most one inhabitant. In the case of the definition above, this means that we will not use the choice of derivation to construct an element of a type.

3. NORMALISATION

Having defined the conversion relation we have specified the system and the goal is to show that conversion is decidable. We are going to establish this by normalisation which has other applications by exploiting the structure of normal forms. Our normal forms are values and partial applications[1] of combinators:

$$\underline{\mathsf{data}}\quad \frac{\sigma\ :\ \mathsf{Ty}}{\mathsf{Nf}_\sigma\ :\ \star}\quad \underline{\mathsf{where}}\quad \frac{}{\mathsf{nK}\ :\ \mathsf{Nf}_{\sigma\to\tau\to\sigma}}\quad \frac{u\ :\ \mathsf{Nf}_\sigma}{\mathsf{nK}^1\,u\ :\ \mathsf{Nf}_{\tau\to\sigma}}$$

$$\frac{}{\mathsf{nS}\ :\ \mathsf{Nf}_{(\sigma\to\tau\to\rho)\to(\sigma\to\tau)\to\sigma\to\rho}}\quad \frac{u\ :\ \mathsf{Nf}_{\sigma\to\tau\to\rho}}{\mathsf{nS}^1\,u\ :\ \mathsf{Nf}_{(\sigma\to\tau)\to\sigma\to\rho}}$$

$$\frac{u\ :\ \mathsf{Nf}_{\sigma\to\tau\to\rho}\quad u'\ :\ \mathsf{Nf}_{\sigma\to\tau}}{\mathsf{nS}^2\,u\,u'\ :\ \mathsf{Nf}_{\sigma\to\rho}}\quad \frac{}{\mathsf{n0}\ :\ \mathsf{Nf}_\mathsf{N}}\quad \frac{}{\mathsf{nsuc}\ :\ \mathsf{Nf}_{\mathsf{N}\to\mathsf{N}}}\quad \frac{n\ :\ \mathsf{Nf}_\mathsf{N}}{\mathsf{nsuc}^1\,n\ :\ \mathsf{Nf}_\mathsf{N}}$$

$$\frac{}{\mathsf{nprec}\ :\ \mathsf{Nf}_{(\mathsf{N}\to\sigma\to\sigma)\to\sigma\to\mathsf{N}\to\sigma}}\quad \frac{u\ :\ \mathsf{Nf}_{\mathsf{N}\to\sigma\to\sigma}}{\mathsf{nprec}^1\,u\ :\ \mathsf{Nf}_{\sigma\to\mathsf{N}\to\sigma}}\quad \frac{u\ :\ \mathsf{Nf}_{\mathsf{N}\to\sigma\to\sigma}\quad u'\ :\ \mathsf{Nf}_\sigma}{\mathsf{nprec}^2\,u\,u'\ :\ \mathsf{Nf}_{\mathsf{N}\to\sigma}}$$

Note that it is an immediate consequence of our definition that the only normal forms of type N are the numerals.

Traditionally, normal forms are considered as a subset of terms — in our setting this is replaced by defining $\ulcorner - \urcorner$ to be a simple embedding from normal forms to terms. It should be noted that the normal forms as subset of terms approach breaks down for more sophisticated systems, e.g. [1].

[1] We use superscripts to indicate how many arguments have already been supplied.

$$\underline{\text{let}} \quad \overline{\ulcorner - \urcorner \;:\; \mathsf{Nf}_\sigma \to \mathsf{Tm}_\sigma}$$

$$\begin{array}{lcl}
\ulcorner \mathsf{nK} \urcorner & \Rightarrow & \mathsf{K} \\
\ulcorner \mathsf{nK}^1\, u \urcorner & \Rightarrow & \mathsf{K} \ulcorner u \urcorner \\
\ulcorner \mathsf{nS} \urcorner & \Rightarrow & \mathsf{S} \\
\ulcorner \mathsf{nS}^1\, u \urcorner & \Rightarrow & \mathsf{S} \ulcorner u \urcorner \\
\ulcorner \mathsf{nS}^2\, u\, u' \urcorner & \Rightarrow & \mathsf{S} \ulcorner u \urcorner \ulcorner u' \urcorner \\
\ulcorner \mathsf{n0} \urcorner & \Rightarrow & \mathsf{0} \\
\ulcorner \mathsf{nsuc} \urcorner & \Rightarrow & \mathsf{suc} \\
\ulcorner \mathsf{nsuc}^1\, n \urcorner & \Rightarrow & \mathsf{suc} \ulcorner n \urcorner \\
\ulcorner \mathsf{nprec} \urcorner & \Rightarrow & \mathsf{prec} \\
\ulcorner \mathsf{nprec}^1\, u \urcorner & \Rightarrow & \mathsf{prec} \ulcorner u \urcorner \\
\ulcorner \mathsf{nprec}^2\, u\, u' \urcorner & \Rightarrow & \mathsf{prec} \ulcorner u \urcorner \ulcorner u' \urcorner
\end{array}$$

Our goal is to define a normalisation function

$$\underline{\text{let}} \quad \overline{\mathbf{nf} \;:\; \mathsf{Tm}_\sigma \to \mathsf{Nf}_\sigma}$$

which should have the following properties:

1. Normalisation takes convertible terms to identical normal forms

$$\frac{a \;\simeq\; a'}{\mathbf{nf}\, a \;=\; \mathbf{nf}\, a'}$$

2. Terms are convertible to their normal forms

$$a \;\simeq\; \ulcorner \mathbf{nf}\, a \urcorner$$

As a consequence we obtain that convertibility corresponds to having the same normal form:

$$t \;\simeq\; u \iff \mathbf{nf}\, t = \mathbf{nf}\, u$$

Since the equality of normal forms is obviously decidable, we have that conversion is decidable. We also obtain that all terms of type N are convertible to a numeral.

As a first approximation we write $\mathbf{nf}'$ and its helper function $\mathbf{napp}'$ to apply normal functions to normal arguments as general recursive functions. The function $\mathbf{napp}'$ uses general recursion, we will show below that it is terminating for all well typed terms — note that the Epigram datatype only contains well typed terms. This seems to be a natural approach: we first implement a recursive function and then show that it is total. We also note that $\mathbf{nf}'$'s type already shows that normalisation is type-preserving, i.e. that a form of subject reduction holds.

$$\underline{\text{let}} \quad \overline{\mathbf{nf}' \;:\; \mathsf{Tm}_\sigma \to \mathsf{Nf}_\sigma}$$

$$\begin{array}{lcll}
\mathbf{nf}' & \mathsf{K} & \Rightarrow & \mathsf{nK} \\
\mathbf{nf}' & \mathsf{S} & \Rightarrow & \mathsf{nS} \\
\mathbf{nf}' & \mathsf{0} & \Rightarrow & \mathsf{n0} \\
\mathbf{nf}' & \mathsf{suc} & \Rightarrow & \mathsf{nsuc} \\
\mathbf{nf}' & \mathsf{prec} & \Rightarrow & \mathsf{nprec} \\
\mathbf{nf}' & (t\, u) & \Rightarrow & \mathbf{napp}'\, (\mathbf{nf}'\, t)\, (\mathbf{nf}'\, u)
\end{array}$$

$$\underline{\text{let}}\quad \frac{}{\mathbf{napp}' \;:\; \mathsf{Nf}_{\sigma\to\tau} \to \mathsf{Nf}_\sigma \to \mathsf{Nf}_\tau}$$

$$\begin{array}{llll}
\mathbf{napp}' & \mathsf{nK} & x & \Rightarrow \quad \mathsf{nK}^1\, x \\
\mathbf{napp}' & (\mathsf{nK}^1\, x) & y & \Rightarrow \quad x \\
\mathbf{napp}' & \mathsf{nS} & x & \Rightarrow \quad \mathsf{nS}^1\, x \\
\mathbf{napp}' & (\mathsf{nS}^1\, x) & y & \Rightarrow \quad \mathsf{nS}^2\, x\, y \\
\mathbf{napp}' & (\mathsf{nS}^2\, x\, y) & z & \Rightarrow \quad \mathbf{napp}'\,(\mathbf{napp}'\, x\, z)\,(\mathbf{napp}'\, y\, z) \\
\mathbf{napp}' & \mathsf{nsuc} & n & \Rightarrow \quad \mathsf{nsuc}^1\, n \\
\mathbf{napp}' & \mathsf{nprec} & f & \Rightarrow \quad \mathsf{nprec}^1\, f \\
\mathbf{napp}' & (\mathsf{nprec}^1\, f) & z & \Rightarrow \quad \mathsf{nprec}^2\, f\, z \\
\mathbf{napp}' & (\mathsf{nprec}^2\, f\, z) & \mathsf{n0} & \Rightarrow \quad z \\
\mathbf{napp}' & (\mathsf{nprec}^2\, f\, z) & (\mathsf{nsuc}^1\, n) & \Rightarrow \quad \mathbf{napp}'\,(\mathbf{napp}'\, f\, n)\,(\mathbf{napp}'\,(\mathsf{nprec}^2\, f\, z)\, n)
\end{array}$$

4. BIG-STEP REDUCTION

We cannot reason about the general recursive functions $\mathbf{nf}'$ and $\mathbf{napp}'$ directly within Type Theory. Instead we are going to specify the graph of these functions as an inductively defined relations a binary relation $-\Downarrow-$ corresponding to $\mathbf{nf}$ and a ternary relation $-\,\$\,-\,\Downarrow-$ coresponding to $\mathbf{napp}$. These relations are precisely the big-step reduction relations. We will use the relations to define termination predicates and we are able to use a variant of [9] to implement terminating versions of the functions.

$$\underline{\text{data}}\quad \frac{t \;:\; \mathsf{Tm}_\sigma \quad n \;:\; \mathsf{Nf}_\sigma}{t \Downarrow n \;:\; \mathsf{Prop}}\quad \underline{\text{where}}$$

$$\frac{}{\mathsf{rK} \;:\; \mathsf{K} \Downarrow \mathsf{nK}} \qquad \frac{}{\mathsf{rS} \;:\; \mathsf{S} \Downarrow \mathsf{nS}} \qquad \frac{p \;:\; t \Downarrow t' \quad q \;:\; u \Downarrow u' \quad r \;:\; t'\;\$\;u' \Downarrow v}{\mathsf{rapp}\, p\, q\, r \;:\; t\, u \Downarrow v}$$

$$\frac{}{\mathsf{r0} \;:\; 0 \Downarrow \mathsf{n0}} \qquad \frac{}{\mathsf{rsuc} \;:\; \mathsf{suc} \Downarrow \mathsf{nsuc}} \qquad \frac{}{\mathsf{rprec} \;:\; \mathsf{prec} \Downarrow \mathsf{nprec}}$$

$$\underline{\text{data}}\quad \frac{m \;:\; \mathsf{Nf}_{\sigma\to\tau} \quad n \;:\; \mathsf{Nf}_\sigma \quad o \;:\; \mathsf{Nf}_\tau}{m\,\$\,n \Downarrow o \;:\; \mathsf{Prop}}\quad \underline{\text{where}}$$

$$\frac{}{\mathsf{rK}^1 \;:\; \mathsf{K}\;\$\;x \Downarrow \mathsf{nK}^1\, x} \qquad \frac{}{\mathsf{rK}^2 \;:\; \mathsf{nK}^1\, x\;\$\;y \Downarrow x} \qquad \frac{}{\mathsf{rS}^1 \;:\; \mathsf{nS}\;\$\;x \Downarrow \mathsf{nS}^1\, x}$$

$$\frac{}{\mathsf{rS}^2 \;:\; \mathsf{nS}^1\, x\;\$\;y \Downarrow \mathsf{nS}^2\, x\, y} \qquad \frac{p \;:\; x\;\$\;z \Downarrow u \quad q \;:\; y\;\$\;z \Downarrow v \quad r \;:\; u\;\$\;v \Downarrow w}{\mathsf{rS}^3\, p\, q\, r \;:\; \mathsf{nS}^2\, x\, y\;\$\;z \Downarrow w}$$

$$\frac{}{\mathsf{rsuc}^1 \;:\; \mathsf{nsuc}\,\$\,n \Downarrow \mathsf{nsuc}^1\, n} \qquad \frac{}{\mathsf{rprec}^1 \;:\; \mathsf{nprec}\;\$\;f \Downarrow \mathsf{nprec}^1\, f} \qquad \frac{}{\mathsf{rprec}^2 \;:\; \mathsf{nprec}^1\, f\;\$\;z \Downarrow \mathsf{nprec}^2\, f\, z}$$

$$\frac{}{\mathsf{rprec0} \;:\; \mathsf{nprec}^2\, f\, z\;\$\;\mathsf{n0} \Downarrow z} \qquad \frac{p \;:\; f\;\$\;n \Downarrow u \quad q \;:\; \mathsf{nprec}^2\, f\, z\;\$\;n \Downarrow v \quad r \;:\; u\;\$\;v \Downarrow w}{\mathsf{rprecsuc}\, p\, q\, r \;:\; \mathsf{nprec}^2\, f\, z\;\$\;\mathsf{nsuc}^1\, n \Downarrow w}$$

We obtain the termination predicates $-\Downarrow$ and $-\,\$\,-\,\Downarrow$ by existentially quantifying over the result:

$$\underline{\text{let}}\quad \frac{t \;:\; \mathsf{Tm}_\sigma}{t \Downarrow \;:\; \mathsf{Prop}} \qquad\qquad \underline{\text{let}}\quad \frac{m \;:\; \mathsf{Nf}_{\sigma\to\tau} \quad n \;:\; \mathsf{Nf}_\sigma}{m\,\$\,n \Downarrow \;:\; \mathsf{Prop}}$$

$$t \Downarrow \quad\Rightarrow\quad \exists n{:}\mathsf{Nf}_\sigma\,.\, t \Downarrow n \qquad\qquad m\,\$\,n \Downarrow \quad\Rightarrow\quad \exists o{:}\mathsf{Nf}_\tau\,.\, m\,\$\,n \Downarrow o$$

We observe that the relations are deterministic — not very surprising since they are derived from function definitions:

Lemma 1

$$\frac{t \Downarrow n \quad t \Downarrow n'}{n = n'} \qquad \frac{f\,\$\,n \Downarrow o \quad f\,\$\,n \Downarrow o'}{o = o'}$$

We have to relate the big-step semantics to our equational theory to be able to show that our normalisation function satisfies the properties 1. and 2. stated above. Property 1 is straightforward and just reflects that our normalisation functions only exploit valid equations:

Lemma 2

$$\frac{t \Downarrow n}{t \simeq \ulcorner n \urcorner} \qquad \frac{f \$ n \Downarrow o}{\ulcorner f \urcorner \ulcorner n \urcorner \simeq \ulcorner o \urcorner}$$

Proof: By induction over the derivations of $t \Downarrow n$ and $f \$ n \Downarrow o$. □

However, property 2., corresponding to confluence, proves more elusive. We would like to have

$$\frac{t \simeq u \quad t \Downarrow n \quad u \Downarrow o}{n = o}$$

However, any proof attempt breaks down at the transitivity rule [2]. Indeed, if our system wouldn't be strongly normalising this property depends on the Church-Rosser property for the small-step semantics, or an equivalent principle. To avoid this complication we anticipate that our big-step semantics is terminating anyway and hence we only need a principle corresponding to the weak Church-Rosser property, which is much easier to show:

Lemma 3 *Assuming normalisation, i.e.* $\forall t{:}\mathsf{Tm}_\sigma \,.\, t \Downarrow$ *we have that*

$$\frac{t \simeq u \quad t \Downarrow n \quad u \Downarrow o}{n = o}$$

Proof: By induction over the derivation of $t \simeq u$, using lemma 1 □

5. STRONG COMPUTABILITY

The essence of Tait's proof was to strengthen strong normalisation to *strong computability* which is, in particular, closed under application. We do the same for normalisation for our bigstep semantics but, slightly misleading, stick to the historic term *Strong Computability*.

We first define a Strong Computability predicate on normal forms SCN by induction over types, here we are using $- \$ - \Downarrow -$ to express that a strongly computable normal form terminates for all strongly computable arguments and produces a strongly computable result:

$$\underline{\text{let}} \quad \frac{t \;:\; \mathsf{Nf}_\sigma}{\mathsf{SCN}_\sigma\, t \;:\; \mathsf{Prop}}$$

$$\begin{array}{lcl} \mathsf{SCN}_\mathsf{N}\, t & \Rightarrow & \textbf{True} \\ \mathsf{SCN}_{\sigma\to\tau}\, t & \Rightarrow & \forall u{:}\mathsf{Nf}_\sigma \,.\, \mathsf{SCN}_\sigma\, u \to \exists n{:}\mathsf{Nf}_\tau \,.\, t \;\$\; u \Downarrow n \wedge \mathsf{SCN}_\tau\, n \end{array}$$

The case for N is so trivial because, as we remarked before, the numerals are exactly the normal forms of type N. In case of more sophisticated inductive types, such as the type of ordinal notations, SCN for that type has to be defined inductively.

A strongly computable term is a term which reduces to a strongly computable normal form — this is represented in Epigram by defining the predicate SC:

$$\underline{\text{let}} \quad \frac{t \;:\; \mathsf{Tm}_\sigma}{\mathsf{SC}_\sigma\, t \;:\; \mathsf{Prop}}$$

$$\mathsf{SC}_\sigma\, t \quad \Rightarrow \quad \exists n{:}\mathsf{Nf}_\sigma \,.\, t \Downarrow n \wedge \mathsf{SCN}_\sigma\, n$$

Our main technical lemma is that all normal forms are strongly computable:

Proposition 1

$$\forall n{:}\mathsf{Nf}_\sigma \,.\, \mathsf{SCN}_\sigma\, n$$

[2] It is interesting to note that transitivity works for call-by-name but then the rule for congruence of application, which is unproblematic for call-by-value, causes trouble.

Proof: We verify the case for a normal forms of arrow type $\sigma \rightarrow \tau$ by induction on the normal form n.

For each case we assume a normal form n' that is strongly computable and we exhibit an inhabitant of the relation $n \,\$\, n' \Downarrow n''$ and show that there is a normal form n'' that is strongly computable.

Case for $\mathsf{nK}^1\, x$:

rK^2 and x is strongly computable by inductive hypothesis.

Case for nK:

rK^1 and $\mathsf{nK}^1\, n$ is strongly computable by the previous case.

Case for $\mathsf{nS}^2\, x\, y$:

We have inductive hypotheses that x and y are strongly computable. It follows that the result of applying each of them to n' to produce normal forms f and a respectively are strongly computable. It then follows that the the application of f to a to the normal form n'' is also strong computable. For the big-step relation we must project out the inhabitants of the relations from the strong computability of f, a and n'' respectively to supply as arguments to rS^3.

Case for $\mathsf{nS}^1\, x$:

rS^2 and $\mathsf{nS}^2\, x\, n$ is strongly computable by the previous case.

Case for nS:

rS^1 and $\mathsf{nS}^1\, n$ is strongly computable by the previous case.

Case for nsuc:

rsuc^1 and $\mathsf{nsuc}^1\, n$ has type Nf_N so is strongly computable.

Case for $\mathsf{nprec}^2\, f\, z$:

For this case we must do a further induction on the argument n which is a natural number. For the $\mathsf{n0}$ case: $\mathsf{rprec0}$ and strong computability of z is by inductive hypothesis. For the $\mathsf{nsuc}^1\, n$ case strong computability is as follows: f is strongly computable by inductive hypothesis, the application of f to n to normal form so f' is strongly computable. $\mathsf{nprec}^2\, f\, z$ is strongly computable by inductive hypothesis and the application of f' to $\mathsf{nprec}^2\, f\, z$ is therefore strongly computable. We exhibit the inhabitant of the reduction relation as follows: The first argument to $\mathsf{rprecsuc}$ is from the strong computability of f' the second is from the inductive hypothesis that $\mathsf{nprec}^2\, f\, z$ is strongly computable and the third from the application of f' to the inductive hypothesis $\mathsf{nprec}^{2\prime}, f\, z$.

Case for $\mathsf{nprec}^1\, f$:

rprec^2 and $\mathsf{nprec}^2\, f\, n$ is strongly computable by the previous case.

Case for nprec:

rprec^1 and $\mathsf{nprec}\, n$ is strongly computable by the previous case. □

The main theorem is now an easy consequence:

Proposition 2 *All terms are Strongly Computable.*

$$\forall t : \mathsf{Tm}_\sigma \,.\, \mathsf{SC}\, t$$

Proof. by induction on t.

Cases for K, S, 0, suc and prec are immediate. Case for application $(t\,u)$: By appeal to the inductive hypotheses we know that t and u have normal forms which are strongly computable. By the definition of SCN their application is strongly computable

Normalisation is an obvious corollary:

Corollary 1 *All terms are normalising.*

$$\underline{\text{let}}\quad \frac{}{\mathbf{norm}\ :\ \forall t{:}\mathsf{Tm}_\sigma\,.\,t\Downarrow}$$

6. IMPLEMENTING NORMALISATION

We have shown our main result — what remains to be done? To implement $\mathbf{nf}$ and $\mathbf{napp}$ in Type Theory we define versions of the functions which take the termination predicates as additional arguments. Here we follow Bove and Capretta, however, the situation is a bit more complicated here because we have nested recursive calls. It has been suggested to use inductive-recursive definitions here [14, 9] but an easier alternative is to use the graph of the function as we have done here. This approach has also been suggested independently by Setzer recently [20].

We first define more general versions of the functions which also return an equation — this is necessary to deal with nested recursion.

$$\underline{\text{let}}\quad \frac{t{:}\mathsf{Tm}_\sigma\quad p{:}t\Downarrow n}{\mathbf{nf}''\,t\,p\ :\ \exists n'{:}\mathsf{Nf}_\sigma\,.\,n'\,=\,n}$$

$$\begin{array}{llll}
\mathbf{nf}'' & \mathsf{K} & \mathsf{rK} & \Rightarrow\ (\mathsf{nK};\ \mathsf{refl})\\
\mathbf{nf}'' & \mathsf{S} & \mathsf{rS} & \Rightarrow\ (\mathsf{nS};\ \mathsf{refl})\\
\mathbf{nf}'' & 0 & \mathsf{r0} & \Rightarrow\ (\mathsf{n0};\ \mathsf{refl})\\
\mathbf{nf}'' & \mathsf{suc} & \mathsf{rsuc} & \Rightarrow\ (\mathsf{nsuc};\ \mathsf{refl})\\
\mathbf{nf}'' & \mathsf{prec} & \mathsf{rprec} & \Rightarrow\ (\mathsf{nprec};\ \mathsf{refl})\\
\mathbf{nf}'' & (t\,u) & (\mathsf{rapp}\,p1\,p2\,p3) & \Rightarrow\ \mathbf{napp}''\,n1\,n2\,p3[\mathbf{coe}\,q1\,q2\rangle\\
\multicolumn{4}{l}{\quad\text{where}\quad (n1;\,q1)=\mathbf{nf}''\,t\,p1}\\
\multicolumn{4}{l}{\qquad\quad (n2;\,q2)=\mathbf{nf}''\,t\,p2}
\end{array}$$

$$\underline{\text{let}}\quad \frac{n{:}\mathsf{Nf}_{\sigma\to\tau},\quad n'{:}\mathsf{Nf}_\sigma\quad p{:}n\ \$\ n'\Downarrow n''}{\mathbf{napp}''\,n\,n'\,p\ :\ \exists n'''{:}\mathsf{Nf}_\tau\,.\,n'''\,=\,n''}$$

$$\begin{array}{llllll}
\mathbf{napp}'' & (\mathsf{nK}) & x & \mathsf{rK}^1 & \Rightarrow & (\mathsf{nK}^1\,x;\ \mathsf{refl})\\
\mathbf{napp}'' & (\mathsf{nK}^1\,x) & y & \mathsf{rK}^2 & \Rightarrow & (x;\ \mathsf{refl})\\
\mathbf{napp}'' & (\mathsf{nS}) & x & \mathsf{rS}^1 & \Rightarrow & (\mathsf{nS}^1\,x;\ \mathsf{refl})\\
\mathbf{napp}'' & (\mathsf{nS}^1\,x) & y & \mathsf{rS}^2 & \Rightarrow & (\mathsf{nS}^2\,x\,y;\ \mathsf{refl})\\
\mathbf{napp}'' & (\mathsf{nS}^2\,x\,y) & z & (\mathsf{rS}^2\,p1\,p2\,p3\,n) & \Rightarrow & \mathbf{napp}''\,n1\,n2\,p3[\mathbf{coe}\,q1\,q2\rangle\\
\multicolumn{6}{l}{\quad\text{where}\quad (n1;\,q1)=\mathbf{napp}''\,x\,z\,p1}\\
\multicolumn{6}{l}{\qquad\quad (n2;\,q2)=\mathbf{napp}''\,y\,z\,p2}\\
\mathbf{napp}'' & (\mathsf{nprec}) & f & \mathsf{rprec}^1 & \Rightarrow & (\mathsf{nprec}^1\,f;\ \mathsf{refl})\\
\mathbf{napp}'' & (\mathsf{nprec}^1\,f) & z & \mathsf{rprec}^2 & \Rightarrow & (\mathsf{nprec}^2\,f\,z;\ \mathsf{refl})\\
\mathbf{napp}'' & (\mathsf{nprec}^2\,f\,z) & \mathsf{n0} & \mathsf{rprec0} & \Rightarrow & (z;\ \mathsf{refl})\\
\mathbf{napp}'' & (\mathsf{nprec}^2\,f\,z) & (\mathsf{nsuc}^1\,n) & (\mathsf{rprecsuc}\,p1\,p2\,p3) & \Rightarrow & \mathbf{napp}''\,n1\,n2\,p3[\mathbf{coe}\,q1\,q2\rangle\\
\multicolumn{6}{l}{\quad\text{where}\quad (n1;\,q1)=\mathbf{napp}''\,f\,n\,p1}\\
\multicolumn{6}{l}{\qquad\quad (n2;\,q2)=\mathbf{napp}''\,(\mathsf{nprec}^2\,f\,z)\,n\,p2}
\end{array}$$

We use the utility function

$$\underline{\text{let}}\quad \frac{p\ :\ m=m'\quad q\ :\ n=n'}{\mathbf{coe}\,p\,q\ :\ m\ \$\ n\Downarrow o=m'\ \$\ n'\Downarrow o}$$

and

$$\underline{\text{let}} \quad \frac{s \ : \ S \quad p \ : \ S = S'}{s[p\rangle \ : \ S'}$$

as introduced in [4]. The definition of $\mathbf{napp}''$ is structurally recursive over the derivation of the big-step relation, if we take into account that coercions are size preserving. This causes some additional effort in the actual formalisation of the construction.

Using the normalisation theorem we can now implement terminating versions of our normalisation function:

$$\begin{array}{l} \underline{\text{let}} \quad \mathbf{nf} \ : \ \mathsf{Tm}_\sigma \to \mathsf{Nf}_\sigma \\ \quad \mathbf{nf} \quad t \quad \Rightarrow \quad n \\ \qquad \text{where} \quad (n;\, -) = \mathbf{nf}''\, t\, (\mathbf{norm}\, t) \end{array}$$

And as a consequence of our previous results we obtain:

Proposition 3 $\mathbf{nf}$ *is a normalisation function, i.e.*

$$\frac{a \ \simeq \ a'}{\mathbf{nf}\, a \ = \ \mathbf{nf}\, a'} \qquad \frac{}{a \ \simeq \ \ulcorner \mathbf{nf}\, a \urcorner}$$

We note that, computationally, the type-theoretic implementation of $\mathbf{nf}$ behaves the same as the recursive implementation since we have only added propositional arguments which cannot affect the computation. This could be made precise by using a compiler based on [10], which eliminates arguments not relevant at run-time.

7. CONCLUSIONS AND FURTHER WORK

It may seem that the last section didn't add much to the result, because indeed proposition 1 already established constructively that for any term there **exists** a normal form which is the result of the big-step reduction. Hence using the first projection for $\exists$ (i.e. by the axiom of choice) we could have obtain a normalisation function directly. However, what is the computational behaviour of this function? As it has already been observed by Berger[7]: we obtain NBE this way — in this case we obtain the normalisation algorithm in [13]. Hence, a potential advantage of the construction proposed here is that we get an implementation of the naive recursive algorithm for normalisation but in a terminating framework. We started with a partial-recursive function and invested some work to obtain the a function with the same computational behaviour but with the guarantuee of termination.

Comparing the technique of normalisation by reduction (NBR) presented here with NBE, which has been applied to the same calculus in [13], we suggest that NBR has the following potential advantages:

- NBR is first order and can be further translated into an abstract machine using standard techniques. NBE requires a higher order meta-language, which already implements functional abstraction.
- NBE is less precise about the actual computational behaviour, i.e. it inherits the evaluation order (e.g. call-by-name vs call-by-value) from the meta-language. It may be useful to vary evaluation order within the same calculus.
- It seems to be often easier to modify the recursively defined normalisation function directly to implement optimisation than to modify the NBE semantics. An example is the admissible equality of $\mathsf{S}\,\mathsf{K}\,\mathsf{K}\,x \ \simeq \ x$, which can be easily added to the recursive normaliser by adding the line

  $$\mathbf{napp}' \quad (\mathsf{nS}^2\,\mathsf{nK}\,\mathsf{nK}) \quad z \quad \Rightarrow \quad z$$

  while the corresponding modification of the NBE semantics is less obvious.
- The separation of proof and algorithm in NBR makes it possible to use a classical normalisation proof, using Markov's principle.

- NBR together with a structural congruence could help to simplify normalisation algorithms for sophisticated systems like λ-calculus with coproducts, for which a normalisation function can be constructed using NBE for a sheaf-theoretic semantics [1].

Having said this, we acknowledge that NBE has a dual collection of advantages, which follows from the fact that it exploits the meta-langauge and hence we don't have to reimplement the basic machinery. The fact that NBE is based on denotational semantics did help to construct normalisation functions, e.g. in the case of coproducts, however, when looking for a better implementation we may have to move to NBR.

We have only dealt with a combinatory calculus and haven't addressed the intricacies of λ-abstraction in the presence of ξ and η rules. However, we do not actually suggest that these extensional rules, should be captured by incorporating them into the big-step relation but by combining a big-step reduction with a structural congruence, which is structurally recursive on types. This approach was suggested in [12] and is indeed the core of the Epigram typechecker as discussed (but not verified) in detail in [11]. We hope that the methodology presented here will be useful when verifying the core of the Epigram system itself.

REFERENCES

[1] Thorsten Altenkirch, Peter Dybjer, Martin Hofmann, and Phil Scott. Normalization by evaluation for typed lambda calculus with coproducts. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 303–310, 2001.

[2] Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Categorical reconstruction of a reduction free normalization proof. In David Pitt, David E. Rydeheard, and Peter Johnstone, editors, *Category Theory and Computer Science*, LNCS 953, pages 182–199, 1995.

[3] Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Reduction-free normalisation for a polymorphic system. In *11th Annual IEEE Symposium on Logic in Computer Science*, pages 98–106, 1996.

[4] Thorsten Altenkirch and Conor McBride. Towards observational type theory. Manuscript, available online, February 2006.

[5] Thorsten Altenkirch, Conor McBride, and James McKinna. Why dependent types matter. Manuscript, available online, April 2005.

[6] Thorsten Altenkirch and Tarmo Uustalu. Normalization by evaluation for $\lambda^{\to 2}$. In *Functional and Logic Programming*, number 2998 in LNCS, pages 260–275, 2004.

[7] Ulrich Berger. Program extraction from normalization proofs. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *LNCS*, pages 91–106. Springer-Verlag, 1993.

[8] Ulrich Berger and Helmut Schwichtenberg. An inverse of the evaluation functional for typed λ–calculus. In R. Vemuri, editor, *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 203–211. IEEE Computer Science Press, Los Alamitos, 1991.

[9] Ana Bove and Venanzio Capretta. Nested general recursion and partiality in type theory. In Richard J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics: 14th International Conference, TPHOLs 2001*, volume 2152 of *Lecture Notes in Computer Science*, pages 121–135. Springer-Verlag, 2001.

[10] Edwin Brady, Conor McBride, and James McKinna. Inductive families need not store their indices. In Stefano Berardi, Mario Coppo, and Ferrucio Damiani, editors, *Types for Proofs and Programs, Torino, 2003*, volume 3085 of *LNCS*, pages 115–129. Springer-Verlag, 2004.

[11] James Chapman, Conor McBride, and Thorsten Altenkirch. Epigram reloaded: A standalone typechecker for ETT. In *Trends in Functional Programming 6*. Intellect, 2005.

[12] Thierry Coquand. An algorithm for testing conversion in type theory. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*. CUP, 1991.

[13] Thierry Coquand and Peter Dybjer. Intuitionistic model constructions and normalization proofs. *Mathematical Structures in Computer Science*, 7(1):75–94, 1997.

[14] Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications*, volume 1581 of *Lecture Notes in Computer Science*, pages 129–146. Springer, April 1999.

[15] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.
[16] Paul Blain Levy. *Call-by-push-value*. PhD thesis, Queen Mary, University of London, 2001.
[17] Conor McBride. Epigram, 2005. `http://www.e-pig.org`.
[18] Conor McBride. Epigram: Practical programming with dependent types. In Varmo Vene and Tarmo Uustalu, editors, *Advanced Functional Programming 2004*, volume 3622 of *Lecture Notes in Computer Science*, pages 130–170. Springer-Verlag, 2005. Revised lecture notes from the International Summer School in Tartu, Estonia.
[19] Conor McBride and James McKinna. The view from the left. *Journal of Functional Programming*, 14(1), 2004.
[20] Anton Setzer. Representation of partial recursive functions by inductive-recursive and by inductive definitions. Talk given at Conference of the Types Project, University of Nottingham, UK, 2006.
[21] W.W. Tait. Intensional interpretations of functionals of finite type. *Journal of Symbolic Logic*, 32:198–212, 1967.