Reduction-free normalisation for a polymorphic system

Thorsten Altenkirch
Ludwig-Maximilians-Universität
Institut für Informatik
Oettingenstr.67, D-80538 München, Germany
E-mail: alti@informatik.uni-muenchen.de

Martin Hofmann, Thomas Streicher
Technische Hochschule Darmstadt
Fachbereich IV (Mathematik)
Schloßgartenstr. 7, D-64289 Darmstadt, Germany
E-mail: {mh|streicher}@mathematik.th-darmstadt.de

Abstract

We give a semantic proof that every term of a combinator version of system F has a normal form. As the argument is entirely formalisable in an impredicative constructive type theory a reduction-free normalisation algorithm can be extracted from this. The proof is presented as the construction of a model of the calculus inside a category of presheaves. Its definition is given entirely in terms of the internal language.

1 Introduction and Summary

In this paper we give a *semantical* proof of *reduction-free normalisation* for F_{SK}, a variant of Girard's system F based on combinators rather than λ-abstraction (for object variables). This generalises the semantical normalisation algorithms for a simply typed systems [2, 5, 1] to polymorphism.

As in loc.cit. we do not prove strong normalisation but construct a function *nf* sending terms to terms in normal form such that convertible terms are sent to the *same* normal form and any term t is convertible with $nf(t)$. Such a function is sufficient for practical purposes as it allows for every term to compute its normal form and thus to decide equality of terms. These "normal forms" are computed by structural induction on terms; no notion of term-rewriting whatsoever is used.

This work is part of a larger programme aiming at deriving reduction-free normalisation algorithms for more complex systems such as Martin-Löf type theory, Extended Calculus of Constructions (ECC), and variants of the Logical Framework. The ultimate goal would be to derive implementations of these systems which would be more efficient than the existing ones because the reduction-free normalisation algorithms can employ the interpreter of the underlying functional programming language as e.g. Standard ML. This gain of efficiency was the initial motivation of Berger and Schwichtenberg for studying reduction-free normalisation. The case of simply-typed lambda calculus has been treated in [2, 5, 1]. The richer systems like ECC extend the simply-typed lambda calculus by two new features: type dependency and polymorphism. We make a first step towards the latter in this paper and leave type dependency for future work.

The key idea of the present work (and also implicit in [5]) is to construct a *model* $\mathcal{G}$ in which types are interpreted as triples $(A, \mathbf{A}^{pred}, \mathbf{A}^{nf})$ where A is a syntactic type, $\mathbf{A}^{pred}$ is a family of "sets" indexed by conversion classes of terms of A, and $\mathbf{A}^{nf}$ is a function mapping an element of $\mathbf{A}^{pred}([t])$ to a normal form in $[t]$, i.e. convertible with t. Unlike in [5] where these "sets" are ordinary sets in our model they are replaced by presheaves over the algebraic theory of types, i.e. contexts and type substitutions.

We summarise our main technical results. Our constructions are carried out within (models of) an impredicative Type Theory, i.e. an extensional version of the Calculus of Constructions extended by inductive types (e.g. similar to [8]). We prove that the category $\hat{\mathbb{S}}$ of constructive-set-valued presheaves over the small category $\mathbb{S}$ of type substitutions forms itself a model of impredicative type theory. We give a characterisation of exponentials in $\hat{\mathbb{S}}$ which allows us to construct a term model of F_{SK} living inside $\hat{\mathbb{S}}$ in which type quantification becomes "set-theoretic" dependent product. The main result consists of the construction of the glued model $\mathcal{G}$ also inside $\hat{\mathbb{S}}$ which can be considered as a naive version of Tait's computability predicates. The desired normalisation function is an immediate corollary of the correctness of this model.

2 The System F_{SK}

We use de Bruijn indices to represent type variables; the judgement $n \vdash A$ means that A is a type with at most n free variables and is defined as follows :

$$\frac{0 \le i < n}{n \vdash i}\ (\textsc{Var}) \qquad \frac{n \vdash A \quad n \vdash B}{n \vdash A \Rightarrow B}\ (\Rightarrow) \qquad \frac{n+1 \vdash A}{n \vdash \forall(A)}\ (\forall)$$

We define the category $\mathbb{S}$ of type substitutions as usual: objects are natural numbers and a morphism in $\mathbb{S}(m,n)$ is an n-tuple of types $m \vdash A_i$. Composition is substitution which can be defined as a recursive function over the syntax. $\mathbb{S}$ has a terminal object 0 and finite products which are given on objects by addition.

If $n+1 \vdash A$ and $n \vdash B$ then we write $A[B]$ for $A \circ \langle id_n, B\rangle$ and B^+ for $B \circ \pi_{n,1}$, where $\pi_n, 1 \in \mathbb{S}(n+1,n)$ is the first projection. We have $n \vdash A[B]$ and $n+1 \vdash B^+$.

The terms together with their types are given by the following rules which define a judgement $n \vdash t : A$ meaning that t is a term with type $n \vdash A$.

$$\frac{n \vdash A \qquad n \vdash B \qquad n \vdash C}{n \vdash \mathsf{s} : (A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow A \Rightarrow C}\ (\mathsf{s})$$

$$\frac{n \vdash A \qquad n \vdash B}{n \vdash \mathsf{k} : A \Rightarrow B \Rightarrow A}\ (\mathsf{k})$$

$$\frac{n \vdash A \qquad n+1 \vdash B}{\begin{array}{c} n \vdash \mathsf{w} : (A \Rightarrow \forall(B)) \Rightarrow \forall(A^+ \Rightarrow B) \\ n \vdash \mathsf{z} : \forall(A^+ \Rightarrow B) \Rightarrow (A \Rightarrow \forall(B)) \end{array}}\ (\mathsf{w},\mathsf{z})$$

$$\frac{n \vdash s : A \Rightarrow B \qquad n \vdash t : A}{n \vdash s\,t : B}\ (\textsc{ap})$$

$$\frac{n+1 \vdash t : A}{n \vdash \Lambda(t) : \forall(A)}\ (\Lambda)$$

$$\frac{n \vdash t : \forall(A) \qquad n \vdash B}{n \vdash t\,B : A[B]}\ (\textsc{Ap})$$

The notion of type substitution can be extended to terms, i.e. given $n+1 \vdash t : A$ and $n \vdash B$ we have $n \vdash t[B] : A[B]$ (more generally: the sets of pairs (t, A) for $n \vdash t : A$ form a presheaf over $\mathbb{S}$).

Equality (convertibility) is given by the least congruence generated by the following equations (we assume implicitly that both sides have the same type):

$$\Lambda(t)\ B = t[B]$$

$$t = \Lambda(t^+\ 0)$$

$$\mathsf{k}\, s\, t = s$$

$$\mathsf{s}\, r\, s\, t = (r\ t)(s\ t)$$

$$\mathsf{w}\, f\ \ C\, s = f\ s\ \ C$$

$$\mathsf{z}\, f\ s\ \ C = f\ C\ s$$

Here A, B, C range over type expressions and f, s, t range over terms.

We write $n \vdash s = t : A$ for $n \vdash s, t : A$ and $s = t$. We remark that we can define functional abstraction in the usual way; the combinators w and z are required to distribute such an abstraction over a type application and abstraction, respectively.

The subset NF of terms in normal form is defined inductively by the following grammar (where $s, t \in NF$ and A, B, C are types):

$$NF ::= \mathsf{k} \mid \mathsf{k}\,t \mid \mathsf{s} \mid \mathsf{s}\,t \mid \mathsf{s}\,s\,t \mid \mathsf{w} \mid \mathsf{w}\,t\,A \mid \mathsf{z} \mid \mathsf{z}\,t \mid \Lambda(t)$$

So the normal forms consist of the combinators partially applied to normal forms and abstractions of normal forms. We do not consider the partial application $\mathsf{w}\,s$ for s a normal form as a normal form because as a term it is equal to $\Lambda(\mathsf{w}\,s^+\,0)$ by rule Eta. Similarly, we do not consider terms of the form $\mathsf{z}\,s\,t$ as a normal form because such term is convertible with $\Lambda X.s\ A\ t$ which according to the form of s may be further reduced.

3 Constructive metalanguage

We understand the previous and the following definitions to be made within a constructive metalanguage which contains an impredicative universe *Prop* of *small sets* closed under inductive definitions. Impredicativity means that the product of an arbitrary family of small sets is again small. Furthermore, we require subset types for equality predicates (i.e. equalisers). A model for such a metalanguage is furnished by ω-Sets and Pers/modest sets as described in [11].

We use an informal (impredicative) extensional Martin-Löf type theory to denote constructions in the metalanguage. In particular, we write Π and Σ for dependent product and sum, and we use λ and juxtaposition for abstraction and application and $\langle -, - \rangle$ for pairing and .1, .2 for the projections. We use the symbol *Type* for the universe of sets in the metalanguage and we form kinds like $A \to \mathit{Type}$ to denote the class of families of sets indexed over A. If $B : A \to \mathit{Type}$ and $f(x), g(x) : B(x)$ if $x{:}\,A$ then we write $\{x{:}\,A \mid f(x) = g(x)\}$ for the subset of A where f and g agree.

Many of our constructions are taking place in specific models of our metalanguage. We overload the syntax of the type theoretic operations but may disambiguate by $\mathit{Type}^C, \Pi^C, \Sigma^C$ where C is the model. Since the models are defined in terms of our constructive metalanguage all internal constructions can be expanded and replaced by much longer (and unreadable) definitions in the metalanguage.

If $a{:}\,A$ and $f(a) = g(a)$ then we may write $a : \{x{:}\,A \mid f(x) = g(x)\}$. Conversely, if $a : \{x{:}\,A \mid f(x) = g(x)\}$ then $a : A$ and furthermore $f(a) = g(a)$ may be inferred. If A is a small set then so is $\{x{:}\,A \mid f(x) = g(x)\}$.

For the definition of functions we either use λ-notation like in $f = \lambda x{:}\,A.t$ or equivalently a "pointwise" notation like in $f(x{:}\,A) = t$. For iterated application we sometimes use parentheses and commas, e.g. if $f : A \to \Pi b{:}\,B.C(b)$ and $a{:}\,A$ and $b{:}\,B$ then we may write $f(a, b)$ instead of $f\ a\ b$. We sometimes omit arguments which can be inferred. For instance, if $f : \Pi a{:}\,A.B(a) \to C$ and $a{:}\,A$ and $b{:}\,B(a)$ then we may abbreviate $f(a, b)$ by $f(b)$.

4 Semantics of F_{SK}

A simple set-theoretic semantics of F_{SK} can be given as follows. We interpret types as small sets, i.e. elements of the impredicative universe *Prop*, and terms as elements. Type quantification is interpreted by impredicative quantification. Rather than spelling out this interpretation in detail we give an abstract notion of model together with a generic interpretation function. The "set-theoretic semantics" then forms a special case.

Definition 1 *An F_{SK}-algebra $\mathcal{A}$ is given by a set $|\mathcal{A}| : \mathit{Type}$ to interpret the types and a family of sets $\mathcal{A} : |\mathcal{A}| \to \mathit{Type}$ to interpret the terms, and functions of the following types :*

$$\begin{array}{rcl}
\Rightarrow & : & |\mathcal{A}| \times |\mathcal{A}| \to |\mathcal{A}| \\
\forall & : & (|\mathcal{A}| \to |\mathcal{A}|) \to |\mathcal{A}| \\
\Lambda & : & \Pi B{:}\,|\mathcal{A}| \to |\mathcal{A}|.(\Pi X{:}\,|\mathcal{A}|.\mathcal{A}(B\,X)) \to \mathcal{A}(\forall\,B) \\
Ap & : & \Pi B{:}\,|\mathcal{A}| \to |\mathcal{A}|.\mathcal{A}\,(\forall\,B) \to \Pi X{:}\,|\mathcal{A}|.\mathcal{A}(B\,X) \\
ap & : & \Pi A, B{:}\,|\mathcal{A}|.\mathcal{A}(A \Rightarrow B) \times \mathcal{A}(A) \to \mathcal{A}(B) \\
\mathsf{s} & : & \Pi A, B, C{:}\,|\mathcal{A}|. \\
 & & \mathcal{A}((A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))) \\
\mathsf{k} & : & \Pi A, B{:}\,|\mathcal{A}|.\mathcal{A}(A \Rightarrow (B \Rightarrow A)) \\
\mathsf{w} & : & \Pi A{:}\,|\mathcal{A}|.\Pi B{:}\,|\mathcal{A}| \to |\mathcal{A}|. \\
 & & \mathcal{A}((A \Rightarrow \forall(B)) \Rightarrow \forall(\lambda X{:}\,|\mathcal{A}|.A \Rightarrow (B\,X))) \\
\mathsf{z} & : & \Pi A{:}\,|\mathcal{A}|.\Pi B{:}\,|\mathcal{A}| \to |\mathcal{A}|. \\
 & & \mathcal{A}(\forall(\lambda X{:}\,|\mathcal{A}|.A \Rightarrow (B\,X))) \Rightarrow (A \Rightarrow \forall(B))
\end{array}$$

Note that we use $\Rightarrow$ in infix notation. $(Ap\ B\ t\ X)$ is abbreviated $t\{X\}$ and $(ap\ A\ B\ s\ t)$ is abbreviated $s^{\imath}t$.

These data are to satisfy the universal closures of the following equations.

$$(\Lambda\,B\,t)\{X\} = t\,X$$

$$\Lambda\,B\,(\lambda X{:}\,|\mathcal{A}|.t\{X\}) = t$$

$$(\mathsf{s}\,A\,B\,C)^{\imath}u^{\imath}v^{\imath}w = (u^{\imath}w)^{\imath}(v^{\imath}w)$$

$$(\mathsf{k}\,A\,B)^{\imath}u^{\imath}v = u$$

$$(\mathsf{w}\,A\,B)^{\imath}u)\{X\}^{\imath}v = (u^{\imath}v)\{X\}$$

$$(((\mathsf{z}\,A\,B)^{\imath}u)^{\imath}v)\{X\} = (u\{X\})^{\imath}v$$

Fix an F_{SK}-algebra $\mathcal{A}$. By induction on derivations we interpret types of F_{SK} in n free variables as functions from $|\mathcal{A}|^n$ to $|\mathcal{A}|$ and terms of type A in n free variables as functions of type $\Pi\vec{X}{:}\,|\mathcal{A}|^n.\mathcal{A}([\![n \vdash A]\!](\vec{X}))$ where $[\![n \vdash A]\!] : |\mathcal{A}|^n \to |\mathcal{A}|$ is the interpretation of the type A. The semantic equations are straightforward: syntactic constructs are interpreted by their semantic counterparts. As examples we give the semantic equations for type quantification and type abstraction:

$$[\![n \vdash \forall(A)]\!] = \lambda\vec{X}{:}\,|\mathcal{A}|^n.\forall(\lambda Y{:}\,|\mathcal{A}|.[\![n+1 \vdash A]\!]\langle\vec{X}, Y\rangle)$$

$$\begin{array}{l}[\![n \vdash \Lambda(t) : \forall(B)]\!] = \\ \qquad \lambda\vec{X}{:}\,|\mathcal{A}|^n.\Lambda(\lambda Y{:}\,|\mathcal{A}|.[\![n+1 \vdash t : B]\!]\langle\vec{X}, Y\rangle)\end{array}$$

Theorem 2 (Soundness) *The interpretation is well-defined on judgements, i.e. does not depend on the derivation chosen. Whenever $n \vdash t = t' : A$ then $[\![n \vdash t : A]\!] = [\![n \vdash t' : A]\!]$.*

The set-theoretic semantics from above can be cast into this abstract framework in the sense that we have an F_{SK}-algebra $\mathcal{P}$ given by $|\mathcal{P}| = \mathit{Prop}$ and $\mathcal{P}(X) = X$. Due to the higher-order nature of the $\forall$- and Λ-operators instances of F_{SK}-algebras do, however, not exist in abundance. We get more instances of F_{SK}-algebras, in particular a term model, if we consider F_{SK}-algebras within the presheaf category $\hat{\mathbf{K}}$ of contravariant functors from some small category $\mathbf{K}$ to the category of (constructive) sets. In particular, we shall define a term model inside the presheaf category $\hat{\mathbb{S}}$.

Before doing this, we explain how presheaf categories form a model of our constructive metalanguage so that all definitions made within this language (in particular the notion of an F_{SK}-algebra) make sense inside such a presheaf category.

5 Presheaves as a model of the constructive metalanguage

Let $\mathbf{K}$ be a small category. It is well-known that the category $\hat{\mathbf{K}} = \mathbf{Set}^{\mathbf{K}^{\mathrm{op}}}$ of presheaves supports extensional Martin-Löf type theory together with inductive definitions even if the ambient set-theoretic universe with respect to which $\hat{\mathbf{K}}$ is formed is not a topos, but only a model of Martin-Löf type theory itself. We refer to [10] for the precise definition of the interpretation of Martin-Löf type theory in a category of presheaves.

We only remind the reader here that dependent types in $\hat{\mathbf{K}}$ can be understood via the equivalence of categories [9, p. 157] $\hat{\mathbf{K}}/F \cong \widehat{El(F)}$ where $El(F) = \mathbf{y}/F$ is the category of elements of the presheaf F. It has as objects pairs (X, f) where $f \in F(X)$ and a morphism from (X, f) to (X', f') is a $\mathbf{K}$-morphism $u : X \to X'$ such that $F(u)(f') = f$. We write $\hat{\mathbf{K}}(F)$ for $\widehat{El(F)}$. So for example the objects of kind $F \to \mathit{Type}^{\hat{\mathbf{K}}}$ are the objects of $\hat{\mathbf{K}}(F)$.

Less well-known is the existence of a small impredicative universe in $\hat{\mathbf{K}}$. Call a presheaf $F \in \hat{\mathbf{K}}$ "small" if for every $X \in \mathbf{K}$ the set $F(X)$ is small, i.e. lies in *Prop*. More generally, if G is any presheaf, we can consider the set $\mathit{Small}(G)$ of small presheaves in $\hat{\mathbf{K}}(G)$. It is obvious that the product of a family of small presheaves is again small because its construction only involves taking products of small sets.

By precomposition the assignment $\mathit{Small}(\_)$ extends to a contravariant set-valued functor on $\hat{\mathbf{K}}$. This functor is representable:

Theorem 3 *The small presheaves over G are in bijective correspondence with $\hat{\mathbf{K}}$-morphisms from G to the presheaf Prop defined by $\mathit{Prop}(X) = \mathit{Small}(\mathbf{y}(X))$ where $\mathbf{y} : \mathbf{K} \to \hat{\mathbf{K}}$ is the Yoneda embedding.*

We shall henceforth identify small presheaves with the associated morphisms into *Prop* and use the informal Martin-Löf type theory explained above in Sect. 3 also to denote constructions in presheaf categories.

5.1 Interpretation of the syntax in an F_{SK}-algebra inside $\hat{\mathbf{K}}$

Assume an F_{SK}-algebra inside some category of presheaves $\hat{\mathbf{K}}$. This means that we have a presheaf $|\mathcal{A}| \in \hat{\mathbf{K}}$ (i.e. $|\mathcal{A}| : \mathit{Type}^{\hat{\mathbf{K}}}$) and a presheaf $\mathcal{A} \in \hat{\mathbf{K}}(|\mathcal{A}|)$ (i.e. $\mathcal{A} : |\mathcal{A}| \to \mathit{Type}^{\hat{\mathbf{K}}}$) and constants as in Def. 1 of the required types which are now understood w.r.t. the internal language of $\hat{\mathbf{K}}$. Again by induction on derivations we can define a semantic function $[\![\_]\!]$ which to $n \vdash A$ associates a $\hat{\mathbf{K}}$-morphism from $|\mathcal{A}|^n$ to $|\mathcal{A}|$ and to terms $n \vdash t : A$ associates a global element of the presheaf $\Pi \vec{X}{:}\, |\mathcal{A}|^n . \mathcal{A}([\![n \vdash A]\!](\vec{X}))$. Viewed externally, $[\![n \vdash A]\!]$ assigns to $I \in \mathbf{K}$ and $\vec{X} \in |\mathcal{A}|(I)^n$ an element $[\![n \vdash A]\!]_I(\vec{X}) \in |\mathcal{A}|(I)$ natural in I and similarly for terms. Again, we have the following soundness property

Proposition 4 *The interpretation of F_{SK} in an F_{SK}-algebra internal to a category of presheaves is sound in the sense that whenever $n \vdash t = t' : A$ then $[\![n \vdash t : A]\!] = [\![n \vdash t' : A]\!]$.*

5.2 The term model as an F_{SK}-algebra

We are now ready to define the desired term model $\mathcal{T}$ internal to $\hat{\mathbb{S}}$. Its object of types $|\mathcal{T}|$ is given by the presheaf *Ty* defined by

$$\begin{aligned}&\mathit{Ty}(n) = \{A \mid n \vdash A\}\\ &\mathit{Ty}(F \in \mathbb{S}(m,n))(A \in \mathit{Ty}(n)) = A[F]\end{aligned}$$

Notice that *Ty* is (isomorphic to) the representable presheaf $\mathbf{y}(1) = \mathbb{S}(\_, 1)$. The object of terms $\mathcal{T} \in \hat{\mathbb{S}}(\mathit{Ty})$ is the presheaf *Tm* defined by

$$\begin{aligned}&\mathit{Tm}(n, A) = \{t \mid n \vdash t : A\}/_{=}\\ &\mathit{Tm}(F)([t]_{=}) = [t[F]]_{=}\end{aligned}$$

Here $=$ refers to judgemental equality of terms, i.e. the set $\mathit{Tm}(n, A)$ consists of equivalence classes of terms modulo convertibility.

For the definition of the remaining components we need the following characterisation of exponentials by representable presheaves.

Lemma 5 *Let* $\mathbf{K}$ *be a category with finite products,* $A \in |\mathbf{K}|$ *be an object of* $\mathbf{K}$*. For any presheaf* $F \in \hat{\mathbf{K}}$ *the exponential* $\mathbf{y}(A) \to F$ *in* $\hat{\mathbf{K}}$ *is isomorphic to the presheaf* $F(- \times A)$*.*

In particular, if $F \in \hat{\mathbb{S}}$ we may identify the presheaf $Ty \to F$ with the presheaf F^+ given by $F^+(n) = F(n+1)$. Accordingly, we define $\forall : (Ty \to Ty) \to Ty$ by $\forall_n(A \in Ty(n+1)) := \forall(A)$. The other components are defined analogously by their syntactic counterparts replacing exponentiation by Ty with "weakening" where appropriate.

Proposition 6 *Let* $[\![-]\!]_{\mathcal{T}}$ *denote the interpretation of the syntax in the term model* $\mathcal{T}$*. We have* $([\![n \vdash A]\!]_{\mathcal{T}})_m(F\colon Ty^n(m)) = A[F]$*, whenever* $n \vdash A$ *and* $([\![n \vdash t : A]\!]_{\mathcal{T}})_m(F\colon Ty^n(m)) = [t[F]]_=$*, whenever* $n \vdash t : A$*.*

5.3 The presheaf of normal forms

The normal forms are stable under generalised substitution meaning that if $n \vdash t : A$ is a normal form and $F \in \mathbb{S}(m,n)$ then $m \vdash t[F] : A[F]$ is also a normal form. This allows us to define a presheaf NF of normal forms over Ty by $NF(n,A) = \{t \in NF \mid n \vdash t : A\}$. We denote by $i : NF \to Tm$ the obvious embedding of normal forms into terms. We remark that i is *a priori* not a monomorphism since elements of Tm are equivalence classes by convertibility.

Proposition 7 *The constructors of the inductive set* NF *give rise to the following functions operating on the presheaf* NF*:*

$\mathsf{k}_0 : \Pi A,B\colon Ty.NF(A \Rightarrow (B \Rightarrow A))$
$\mathsf{k}_1 : \Pi A,B\colon Ty.NF(A) \to NF(B \Rightarrow A)$
$\mathsf{s}_0 : \Pi A,B,C\colon Ty.NF((A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow C))$
$\mathsf{s}_1 : \Pi A,B,C\colon Ty.NF(A \Rightarrow (B \Rightarrow C)) \to NF((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$
$\mathsf{s}_2 : \Pi A,B,C\colon Ty.NF(A \Rightarrow (B \Rightarrow C)) \to NF(A \Rightarrow B) \to NF(A \Rightarrow C)$
$\mathsf{w}_0 : \Pi A\colon Ty.\Pi B\colon Ty \to Ty.NF((A \Rightarrow \forall(B)) \Rightarrow \forall(\lambda X\colon Ty.A \Rightarrow (B\,X)))$
$\mathsf{w}_2 : \Pi A\colon Ty.\Pi B\colon Ty \to Ty.NF(A \Rightarrow \forall(B)) \to \Pi X\colon Ty.NF(A \Rightarrow (B\,X))$
$\mathsf{z}_0 : \Pi A\colon Ty.\Pi B\colon Ty \to Ty.NF(\forall(\lambda X\colon Ty.A \Rightarrow (B\,X)) \Rightarrow (A \Rightarrow \forall(B)))$
$\mathsf{z}_1 : \Pi A\colon Ty.\Pi B\colon Ty \to Ty.NF(\forall(\lambda X\colon Ty.A \Rightarrow (B\,X))) \to NF(A \Rightarrow \forall(B))$
$\Lambda_{NF} : \Pi B\colon Ty \to Ty.(\Pi X\colon Ty.NF(B\,X)) \to NF(\forall\,B)$

The universal closures of the following equations are valid where the right hand sides refer to the term model $\mathcal{T}$*.*

$$\begin{aligned}
&i(\mathsf{k}_0(A,B)) = \mathsf{k}(A,B)\\
&i(\mathsf{k}_1(A,B,t)) = \mathsf{k}(A,B)^{\iota}i(t)\\
&i(\mathsf{s}_0(A,B,C)) = \mathsf{s}(A,B,C)\\
&i(\mathsf{s}_1(A,B,C,t)) = \mathsf{s}(A,B,C)^{\iota}i(t)\\
&i(\mathsf{s}_2(A,B,C,s,t)) = \mathsf{s}(A,B,C)^{\iota}i(s)^{\iota}i(t)\\
&i(\mathsf{w}_0(A,B)) = \mathsf{w}(A,B)\\
&i(\mathsf{w}_2(A,B,t,X)) = \mathsf{w}(A,B)^{\iota}i(t)\{X\}\\
&i(\mathsf{z}_0(A,B)) = \mathsf{z}(A,B)\\
&i(\mathsf{z}_1(A,B,t)) = \mathsf{z}(A,B)^{\iota}i(t)\\
&i(\Lambda_{NF}(B,t)) = \Lambda(\lambda X\colon Ty.i(t\,X))
\end{aligned}$$

6 The glued model

Our aim here is to define an F_{SK}-algebra internal to the arrow category $\hat{\mathbb{S}}^{\to}$ in which syntactic terms are "glued" together with functions describing how they act on normal forms. By the isomorphism $(\mathbf{Set}^{\mathbb{S}^{\mathrm{op}}})^{\to} \cong \mathbf{Set}^{\mathbb{S}^{\mathrm{op}} \times \to}$, where $\to$ is the category with two objects $0, 1$ and one non-identity morphism from 0 to 1 the arrow category $\hat{\mathbb{S}}^{\to}$ is also a category of presheaves and so the general theory applies.

We shall, however, only make use of the internal language of $\hat{\mathbb{S}}$ and construct the required objects in $\hat{\mathbb{S}}^{\to}$ explicitly. To that end it is appropriate to employ the equivalence between families of types and their associated display maps. More precisely, we view an object of $\hat{\mathbb{S}}^{\to}$ as a pair (I,X) where $I \in \hat{\mathbb{S}}$ and $X : I \to Type^{\hat{\mathbb{S}}}$, i.e. $X \in \hat{\mathbb{S}}(I)$. Accordingly, a morphism from object (I,X) to object (J,Y) in this presentation consists of a function $f : I \to J$ and a function $\mathbf{f} : \Pi i\colon I.X(i) \to Y(f\,i)$. The exponential of (I,X) and (J,Y) in $\hat{\mathbb{S}}^{\to}$ is then given by the pair $I \to J$ and $\lambda f\colon I \to J.\Pi i\colon I.X(i) \to Y(f\ i)$. Notice the obvious similarity with Logical Predicates and 'deliverables'. If (I,X) is an object of $\hat{\mathbb{S}}^{\to}$ presented as a 'deliverable' then a *family* (in $\hat{\mathbb{S}}^{\to}$) over (I,X) is a pair (J,Y) where $J : I \to Type$ and $Y : \Pi i\colon I.J(i) \times X(i) \to Type$. The dependent product of such a family is given by the pair $\Pi i\colon I.J(i)$ and $\lambda f\colon \Pi i\colon I.J(i).\Pi i\colon I.\Pi x\colon X(i).Y(i, f\,i, x)$.

6.1 Types and terms in the glued model

We can now construct the desired glued model $\mathcal{G}$ internal to $\hat{\mathbb{S}}^{\to}$. Its first component will be identical to the term model, i.e. the functor $Fst : \hat{\mathbb{S}}^{\to} \to \hat{\mathbb{S}}$ sending (I,X) to I maps the glued model to the term model. For this reason we refer to the first component I of an object (I,X) as the *syntactic* component and to the second one (X) as the *semantic* component of (I,X). We use the same terminology for morphisms (and elements). If A is an object of $\mathcal{G}$ we refer to its components by A^{syn} and A^{sem}, respectively, and so we do for morphisms.

Accordingly, the object of types in the glued model takes the form $|\mathcal{G}| = (Ty, \mathbf{Ty})$ where $\mathbf{Ty} : Ty \to Type$ is given by

$$\begin{aligned} &\mathbf{Ty}(A{:}\, Ty) = \\ &\quad \Sigma P{:}\, Tm(A) \to Prop.\Pi t{:}\, Tm(A). \\ &\quad P(t) \to \{t'{:}\, NF(A) \mid (i\, t') = t\} \end{aligned}$$

So an element of $\mathbf{Ty}(A)$ consists of a "proof-relevant predicate on $Tm(A)$", i.e. a small presheaf P indexed over $Tm(A)$ and for every term t and element of $P(t)$ (a "proof" of $P(t)$) a normal form t' which is convertible to term t. If $\mathbf{A} : \mathbf{Ty}(A)$ we refer to its components by $\mathbf{A}^{pred}$ and $\mathbf{A}^{nf}$, i.e. we have $\mathbf{A}^{pred} : Tm(A) \to Prop$ and $\mathbf{A}^{nf} : \Pi t{:}\, Tm(A).\mathbf{A}^{pred}(t) \to NF(A)$ and if $t{:}\, Tm(A)$ and $\mathbf{t}{:}\, \mathbf{A}^{pred}(t)$ then $i(\mathbf{A}^{nf}(t, \mathbf{t})) = t$. We also abbreviate $\mathbf{A}^{nf}(t, \mathbf{t})$ by $\mathtt{q}(\mathbf{t})$ if t and $\mathbf{A}$ are clear from the context. The property "has a normal form" itself forms an element of $\mathbf{Ty}(X)$ for every $X{:}\, Ty$, more precisely, we define $\mathtt{U} : \Pi X{:}\, Ty.\mathbf{Ty}(X)$ by:

$$\begin{aligned} \mathtt{U}(X)^{pred} &= \lambda t{:}\, Tm(X).\{t'{:}\, NF(X) \mid i(t') = t\} \\ \mathtt{U}(X)^{nf} &= \lambda t{:}\, Tm(X).\lambda \mathbf{t}{:}\, \mathtt{U}(X)^{pred}(t).\mathbf{t} \end{aligned}$$

The object of terms $\mathcal{G}$ takes the form $(Tm, \mathbf{Tm})$ where $\mathbf{Tm} : \Pi A{:}\, Ty.\mathbf{Ty}(A) \times Tm(A) \to Type$ is defined by $\mathbf{Tm}(A, \mathbf{A}, t) = \mathbf{A}^{pred}(t)$. It remains to define the required functions on types and terms and to establish the equations.

6.2 Interpretation of type quantification

An element of $|\mathcal{G}| \to^{\hat{\mathbb{S}}\to} |\mathcal{G}|$ is given by $B : Ty \to Ty$ (the syntactic component) and $\mathbf{B} : \Pi X{:}\, Ty.\mathbf{Ty}(X) \to \mathbf{Ty}(B\, X)$ (the semantic component). From these data we have to construct $\forall(B, \mathbf{B}) : \mathbf{Ty}(\forall(B))$ [1]

$$\begin{aligned} &\forall(B, \mathbf{B})^{pred}(t) = \\ &\quad \Pi X{:}\, Ty.\Pi \mathbf{X}{:}\, \mathbf{Ty}(X).\mathbf{B}(X, \mathbf{X})^{pred}(t\{X\}) \\ &\forall(B, \mathbf{B})^{nf}(t, \mathbf{t}) = \\ &\quad \Lambda_{NF}(\lambda X{:}\, Ty.\mathtt{q}((\mathbf{t}(X, \mathtt{U}(X))))) \end{aligned}$$

where $t : Tm(\forall(B)), \mathbf{t} : \forall(\mathbf{B})^{pred}(t)$.

Notice that impredicativity of $Prop \in \hat{\mathbb{S}}$ is required in this place. It is straightforward to verify that the equation $i(\forall(B, \mathbf{B})^{nf}(\mathbf{t})) = t$ holds.

For the abstraction assume $t : \Pi X{:}\, Ty.Tm(B\ X)$ and

$$\mathbf{t} : \Pi X{:}\, Ty.\Pi \mathbf{X}.\mathbf{Ty}(X).\mathbf{B}(X, \mathbf{X}, t\, X).$$

[1]Note that the syntactic component is already fixed by the requirement that Fst induces a homomorphism from $\mathcal{G}$ to $\mathcal{T}$

Now

$$\begin{aligned} &\mathbf{Tm}(\forall(B), \forall(B, \mathbf{B}), \Lambda(t)) = \\ &\quad \Pi X{:}\, Ty.\Pi \mathbf{X}{:}\, \mathbf{Ty}(X).\mathbf{B}(X, \mathbf{X})^{pred}(\Lambda(t)\{X\}) \end{aligned}$$

by definition. The latter equals

$$\Pi X{:}\, Ty.\Pi \mathbf{X}{:}\, \mathbf{Ty}(X).\mathbf{B}(X, \mathbf{X})^{pred}(t\, X)$$

Thus we have

$$\mathbf{t} : \mathbf{Tm}(\forall(B), \forall(B, \mathbf{B}), \Lambda(t))$$

and we can define

$$\mathbf{\Lambda}(B, \mathbf{B}, t, \mathbf{t}) := \mathbf{t}$$

For application assume that $t : Tm(\Pi(B))$ and $\mathbf{t} : \mathbf{Tm}(\forall(B), \forall(B, \mathbf{B}), t)$. Like before we can show that

$$\mathbf{Tm}(\forall(B), \forall(B, \mathbf{B}), t) = \Pi X{:}\, Ty.\Pi \mathbf{X}{:}\, \mathbf{Ty}(X).\mathbf{Tm}(BX, \mathbf{B}(X, \mathbf{X}), t\{X\}$$

So for $X{:}\, Ty$ and $\mathbf{X}{:}\, \mathbf{Ty}(X)$, we define

$$\mathbf{Ap}(B, \mathbf{B}, t, \mathbf{t}, X, \mathbf{X}) = \mathbf{t}(X, \mathbf{X}) : \mathbf{Tm}(BX, \mathbf{B}(X, \mathbf{X}), t\{X\})$$

The semantic equations can be seen to hold simply by expanding the definitions of $\mathbf{\Lambda}$ and $\mathbf{Ap}$.

6.3 Interpretation of function types

Assume that $A, B{:}\, Ty$ and $\mathbf{A}{:}\, \mathbf{Ty}(A)$ and $\mathbf{B}{:}\, \mathbf{Ty}(B)$. We have to define the semantic component $\mathbf{A} \Rightarrow \mathbf{B} : \mathbf{Ty}(A \Rightarrow B)$. This is done as follows:

$$\begin{aligned} &(\mathbf{A} \Rightarrow \mathbf{B})^{pred}(t{:}\, Tm(A \Rightarrow B)) = \\ &\quad \{t'{:}\, NF(A \Rightarrow B) \mid i(t') = t\} \times \\ &\quad \Pi a{:}\, Tm(A).\mathbf{Tm}(A, \mathbf{A}, a) \to \mathbf{Tm}(B, \mathbf{B}, t^{l}a) \end{aligned}$$

$$(\mathbf{A} \Rightarrow \mathbf{B})^{nf}(t{:}\, Tm(A \Rightarrow B)\, ,\, \mathbf{t}{:}\, (\mathbf{A} \Rightarrow \mathbf{B})^{pred}(t)) = \mathbf{t}.1$$

Intuitively, $t : Tm(A \Rightarrow B)$ "satisfies" $\mathbf{A} \Rightarrow \mathbf{B}$ if it is convertible to a normal form and sends objects satisfying $\mathbf{A}^{pred}$ to objects satisfying $\mathbf{B}^{pred}$.

As for term application we start from $u{:}\, Tm(A \Rightarrow B)$ and $v{:}\, Tm(A)$ together with $\mathbf{u}{:}\, \mathbf{Tm}(A \Rightarrow B, \mathbf{A} \Rightarrow \mathbf{B}, u)$ and $\mathbf{v}{:}\, \mathbf{Tm}(A, \mathbf{A}, v)$. We have $\mathbf{u}.2(v, \mathbf{v}) : \mathbf{Tm}(B, \mathbf{B}, u^{l}v)$. This suggests to define the semantic component of application by

$$\mathbf{ap}(A, \mathbf{A}, B, \mathbf{B}, u, \mathbf{u}, v, \mathbf{v}) = \mathbf{u}.2(v, \mathbf{v}),$$

which works as required.

It remains to define the combinators. Their definition follows closely the development in [5] except that we work in the internal language of $\hat{\mathbb{S}}$ here. For the k-combinator asssume $A, B{:}\, Ty$ and $\mathbf{A}{:}\, \mathbf{Ty}(A)$ and

$\mathbf{B}\colon \mathbf{Ty}(B)$ as before. The semantic component $\mathbf{k}$ of the k-combinator is an element of

$$\mathbf{Tm}(A \Rightarrow (B \Rightarrow A)\,,\, (\mathbf{A} \Rightarrow (\mathbf{B} \Rightarrow \mathbf{A}))\,,\, \mathsf{k})$$

We define it by

$$\begin{aligned}\mathbf{k}(A, \mathbf{A}, B, \mathbf{B}) = \langle & \mathsf{k}_0(A, B), \lambda a\colon Tm(A).\lambda \mathbf{a}\colon \mathbf{Tm}(\mathbf{A}, a).\\ & \langle \mathsf{k}_1(A, B, \mathsf{q}(\mathbf{a})),\\ & \quad \lambda b\colon Tm(B).\lambda \mathbf{b}\colon \mathbf{Tm}(\mathbf{B}, b).\mathbf{a}\rangle\rangle\end{aligned}$$

To see that this is type correct we observe that in its context $\mathbf{a}$ has type $\mathbf{Tm}(\mathbf{A}, \mathsf{k}\wr a \wr b)$ because the latter equals $\mathbf{Tm}(\mathbf{A}, a)$ by the conversion rule for k. Again, the semantic equation for k, i.e. the universal closure of $\mathbf{ap}(\mathbf{ap}(\mathbf{k}(A, \mathbf{A}, B, \mathbf{B}), \mathbf{a}), \mathbf{b}) = \mathbf{a}$ follows easily by equality reasoning.

For the remaining combinators s, w, z we follow the same pattern: we use the partially applied normal forms ($\mathsf{s}_0, \mathsf{w}_0$, etc.) as witnesses of the required normal forms and the proof objects that the subterms occurring in the construction satisfy the predicates are constructed by mimicking the combinators on the semantic level. For the ease of the reader we give the definitions of the semantic combinators here.

$$\begin{aligned}&\mathbf{s}(A, \mathbf{A}, B, \mathbf{B}, C, \mathbf{C}) =\\ &\quad \langle \mathsf{s}_0(A, B, C),\\ &\quad\ \lambda u\colon Tm(A \Rightarrow B \Rightarrow C).\lambda \mathbf{u}\colon \mathbf{Tm}(\mathbf{A} \Rightarrow \mathbf{B} \Rightarrow \mathbf{C}, u).\\ &\qquad \langle \mathsf{s}_1(A, B, C, \mathsf{q}(\mathbf{u})),\\ &\qquad\ \lambda v\colon Tm(A \Rightarrow B).\lambda \mathbf{v}\colon \mathbf{Tm}(\mathbf{A} \Rightarrow \mathbf{B}, v).\\ &\qquad\quad \langle \mathsf{s}_2(A, B, C, \mathsf{q}(\mathbf{u}), \mathsf{q}(\mathbf{v})),\\ &\qquad\quad\ \lambda w\colon Tm(A).\lambda \mathbf{w}\colon \mathbf{Tm}(\mathbf{A}, w)\,.\\ &\qquad\qquad (\mathbf{u}.2\ \mathbf{w}).2\ (\mathbf{v}.2\ \mathbf{w})\rangle\rangle\rangle\end{aligned}$$

$$\begin{aligned}&\mathbf{w}(A, \mathbf{A}, B, \mathbf{B}) =\\ &\quad \langle \mathsf{w}_0(A, B),\\ &\quad\ \lambda u\colon Tm(A \Rightarrow \forall(B)).\lambda \mathbf{u} : \mathbf{Tm}(\mathbf{A} \Rightarrow \forall(\mathbf{B})).\\ &\qquad \lambda X\colon Ty.\lambda \mathbf{X}\colon \mathbf{Ty}(X).\\ &\qquad\quad \langle \mathsf{w}_2(A, B, \mathsf{q}(\mathbf{u}), X),\\ &\qquad\quad\ \lambda a\colon Tm(A).\lambda \mathbf{a}\colon \mathbf{Tm}(\mathbf{A}, a).\mathbf{u}.2(\mathbf{a}, \mathbf{X})\rangle\rangle\end{aligned}$$

$$\begin{aligned}&\mathbf{z}(A, \mathbf{A}, B, \mathbf{B}) =\\ &\quad \langle \mathsf{z}_0(A, B),\\ &\quad\ \lambda u\colon Tm(\forall(\lambda X\colon Ty.A \Rightarrow (BX))).\\ &\quad\ \lambda \mathbf{u}\colon \mathbf{Tm}(\forall(\lambda X\colon Ty.\lambda \mathbf{X}\colon \mathbf{Ty}(X).\mathbf{A} \Rightarrow \mathbf{B}(\mathbf{X}))).\\ &\qquad \langle \mathsf{z}_1(A, B, \mathsf{q}(\mathbf{u})),\\ &\qquad\ \lambda a\colon Tm(A).\lambda \mathbf{a}\colon \mathbf{Tm}(\mathbf{A}, a).\\ &\qquad\ \lambda X\colon Ty.\lambda \mathbf{X}\colon \mathbf{Ty}(X).\mathbf{u}.2(\mathbf{X}, \mathbf{a})\rangle\rangle\end{aligned}$$

This completes the definition and verification of the glued model $\mathcal{G}$.

7 The normalisation function

Following the pattern already exhibited in [5, 1] we can now extract a normalisation function from the interpretation of the syntax in the glued model. Let $[\![-]\!]_{\mathcal{G}}$ and $[\![-]\!]_{\mathcal{T}}$ denote the interpretations in $\mathcal{G}$ and $\mathcal{T}$, respectively. Recall from Section 5.1 that $[\![-]\!]^{syn}_{\mathcal{G}}$ and $[\![-]\!]_{\mathcal{T}}$ are natural transformations in $\hat{\mathbb{S}}$ the fibres of which we denote by $[\![-]\!]^{syn}_{\mathcal{G},m}$ and $[\![-]\!]_{\mathcal{T},m}$ for $m \in \mathbb{N}$.

Proposition 8 *Assume $m, n \in \mathbb{N}$, $F \in \mathbb{S}(m, n) = Ty^n(m)$, and $n \vdash A$ and $n \vdash t : A$. We have*

$$[\![n \vdash A]\!]^{syn}_{\mathcal{G},m}(F) = [\![n \vdash A]\!]_{\mathcal{T},m}(F) = A[F]$$

and also

$$[\![n \vdash t : A]\!]^{syn}_{\mathcal{G},m}(F) = [\![n \vdash t : A]\!]_{\mathcal{T},m}(F) = [t[F]]_{=}$$

Proof. Immediate from the definition of $\mathcal{G}$ and Prop. 6. □

Theorem 9 *For every closed type $0 \vdash A$ there exists a constructively definable function nf_A mapping closed terms of type A to normal forms in such a way that if $0 \vdash t : A$ then $0 \vdash nf_A(t) = t : A$ and if $0 \vdash t = t' : A$ then $nf_A(t)$ and $nf_A(t')$ are syntactically equal.*

Proof. For $0 \vdash t : A$ we make the definitions $\mathbf{A} := [\![0 \vdash A]\!]^{sem}$ and $\mathbf{t} := [\![0 \vdash t : A]\!]^{sem}$. We define

$$nf_A(t) := \mathbf{A}^{nf}{}_0(\mathbf{t}_0(\star))$$

If $0 \vdash t = t' : A$ then $[\![0 \vdash t : A]\!] = [\![0 \vdash t' : A]\!]$ by soundness and hence $nf_A(t) \equiv nf_A(t')$. To see that $0 \vdash t = nf_A(t) : A$ consider the following diagram[2] instantiated at 0 which commutes by definition of the glued model.

$$\begin{tikzcd} & \mathbf{Tm}(\mathbf{A}, \_) \arrow[d] \arrow[dr, "\mathbf{A}^{nf}"] & \\ \mathbf{1} \arrow[ur, "\mathbf{t}"] \arrow[r, "{[\![0 \vdash t : A]\!]^{syn}}"'] & Tm([\![0 \vdash A]\!]^{syn}) & NF(A) \arrow[l, "i"] \end{tikzcd}$$

It expresses that that $i(nf(t)) = i(\mathbf{A}^{nf}{}_0(\mathbf{t}_0(\star))) = [\![0 \vdash t : A]\!]^{syn}_0(\star)$. By Prop. 8 the latter equals $[t]_{=}$, i.e. the equivalence class of t under conversion. □

The restriction to closed terms is immaterial as by type abstraction the closed terms are in 1-1 correspondence to the open ones.

[2]This diagram has been typeset using Paul Taylor's Latex package.

8 Conclusion and further work

We have given a Tait-style proof of normalisation for a combinator version of system F. By working inside a category of presheaves we were able to consider type quantification as a higher-order operation which allows for particularly simple clauses in the definition of the "computability predicates". We emphasise that our proof includes η-conversion on the level of types. In verifying the construction of $\mathcal{G}$ from the term model we have successfully employed the *Lego* system as a proof assistant for the internal language of $\hat{\mathbb{S}}$.

Our method can be extended to full system F based on functional abstraction instead of combinators and including the corresponding η-rule. This can be achieved by working in the category of presheaves over type substitutions and weakenings of ordinary contexts (cf. [1]). In this category the term model takes the form of an internal CCC closed under *Ty*-indexed products. Again, the glued model has the same form, but as before lives in the corresponding arrow category. A detailed exposition is in preparation.

A by-product of our work consists of a semantic justification of *higher-order abstract syntax* [7, 6]. We have shown how type quantification (even in the syntax) arises naturally as a higher-order operation when working inside a category of presheaves. In the presheaf category to be used for full system F we can also consider functional abstraction as a higher-order operation. According to whether we use full substitutions or merely weakenings as the site of definition we obtain the type $(Tm(A) \to Tm(B)) \to Tm(A \Rightarrow B)$ or $(\mathrm{Par}(A) \to Tm(B)) \to Tm(A \Rightarrow B)$ as the type of functional abstraction. Here $\mathrm{Par}(A)$ is the semantical counterpart to "variables" of type A as given by the presheaf represented by A. The latter may be compared to the approach described in [6]. This aspect might be of interest in its own.

In order to effectively extract a program for normalisation one would have to produce a formalisation which actually implements presheaves and natural transformations and not merely refers to them abstractly via the internal language of $\hat{\mathbb{S}}$. All the definitions which for ease of exposition we have made in terms of the internal language have to be expanded into their set-theoretic or rather type-theoretic formulations. There is no principle obstacle in doing so, but obviously such formalisation would be very laborious. For a simply-typed system this has effectively been carried out in [4].

From such a formalisation one can immediately derive an executable function *nf*. This function *nf* is given by type-theoretic code which can be seen as a functional program.

To achieve reasonable performance this code can be transformed into an SML program (see Appendix 8). To do this one has to systematically replace dependently-typed inductive definitions by recursive datatypes and delete computationally-irrelevant arguments.

For example the type of the semantic values which in the type-theoretic formulation depends on types and on (convertibility classes of) terms becomes in Standard ML the following recursive datatype where and `Nf` is a simple inductive definition of normal forms.

```
datatype Sem = u of Nf
             | s_arr of Nf*(Sem -> Sem)
             | s_pi of (Sem -> Nf) -> Sem;
```

The three constructors correspond to the three possibilities of forming semantic objects explained in Sections 6.1, 6.2, and 6.3.

In order to justify these changes of data structure a general theory is called for. Lacking such theory at the moment we have to content ourselves with intuition and—if desired—a brute-force verification of the SML program using logical relations defining in which sense the SML program simulates the type-theoretic one.

A An SML program extracted from the construction

```
infixr -->;

datatype Ty = var of int |
            | pi of Ty
            | --> of Ty*Ty;

datatype Tm = s
            | k of Ty
            | w
            | z of Ty
            | app of Tm*Tm
            | Lam of Tm
            | App of Tm*Ty;

datatype Nf = s0 | s1 of Nf | s2 of Nf*Nf
            | k0 | k1 of Nf
            | w0 | w1 of Nf
            | z0 | z1 of Nf
            | Lam' of Nf;


datatype Sem = s_u of Nf
           | s_arr of Nf*(Sem -> Sem)
           | s_pi of (Sem -> Nf) -> Sem;

fun s_u' (s_u M) = M;
fun sem_app (s_arr (M,f)) = f;

fun sem_App (s_pi f) = f;

fun quote (var i)   qs  x =
      nth(qs,i)(x)
  | quote (_ --> _) qs (s_arr (M,_)) =
      M
  | quote (pi A)    qs (s_pi f) =
      Lam' (quote A (s_u'::qs) (f s_u'))

fun eval (k(A)) qs =
     s_arr (k0,fn x =>
      s_arr (k1(quote A qs x),fn y => x))
  | eval s      qs =
      s_arr (s0,fn (s_arr(M,f)) =>
        s_arr(s1(M),fn (s_arr (N,g)) =>
          s_arr(s2(M,N),fn x =>
    sem_app (f x) (g x))))
  | eval w      qs =
      s_arr (w0,fn (s_arr (M,f)) =>
        s_pi (fn q =>
              (s_arr (w1(M),fn x => f x))))
  | eval (z(A)) qs =
      s_arr (z0,fn f =>
             s_arr (z1(quote A qs f),fn x =>
               s_pi (fn q =>
                sem_app (sem_App f q) x)))
  | eval (app (M,N)) qs =
      sem_app (eval M qs) (eval N qs)
  | eval (Lam M) qs =
      s_pi (fn q => eval M (q::qs))
  | eval (App (M,A)) qs =
      sem_App (eval M qs) (quote A qs)

fun nf M A = quote A [] (eval M []);
```

Examples:

```
val poly_id = Lam (app(app(s,k(var 0)),k(var 0)))
val unit = (pi ((var 0) --> (var 0)));
nf poly_id unit;
nf (App(poly_id,unit)) (unit --> unit);
nf (app(App(poly_id,unit),poly_id)) unit;
```

References

[1] T. Altenkirch, M. Hofmann, and T. Streicher. Categorical reconstruction of a reduction-free normalisation proof. In D. Pitt and D. Rydeheard, editors, *Proc. CTCS '95, Springer LNCS, Vol. 953*, pages 182–199, 1995.

[2] U. Berger and H. Schwichtenberg. An inverse of the evaluation functional for typed λ-calculus. In *Proceedings of LICS '91, Amsterdam*, pages 203–211, 1991.

[3] R. Constable et al. *Implementing Mathematics with the Nuprl Development System.* Prentice-Hall, 1986.

[4] C. Coquand. From semantics to rules: a machine-assisted analysis. In Börger, Gurevich, and Meinke, editors, *Proceedings of CSL '93*, pages 171–185. Springer, LNCS, 1994.

[5] T. Coquand and P. Dybjer. Intuitionistic model constructions and normalization proofs. *Mathematical Structures in Computer Science*, 1993. to appear, previous version has appeared in the informal proceedings of the BRA-Types workshop, 1993 held in Turin.

[6] J. Despeyroux, A. Felty, and A. Hirschowitz. Higher-order abstract syntax in Coq. In M. Dezani and G. Plotkin, editors, *Typed Lambda Calculi and Applications*, pages 124–138. Springer LNCS vol. 902, 1995.

[7] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, Jan. 1993.

[8] Z. Luo. *Computation and Reasoning.* Oxford University Press, 1994.

[9] I. Moerdijk and S. M. Lane. *Sheaves in Geometry and Logic. A First Introduction to Topos Theory.* Springer, 1992.

[10] W. Phoa. An introduction to fibrations, topos theory, the effective topos, and modest sets. Technical Report ECS-LFCS-92-208, LFCS Edinburgh, 1992.

[11] T. Streicher. *Semantics of Type Theory.* Birkhäuser, 1991.