

Strong Normalization for Church-Style System F

Arthur Freitas Ramos David Barros Hulak
Ruy J. G. B. de Queiroz

October 6, 2026

Contents

1 Overview

1

Abstract

This entry formalizes strong normalization for Church-style System F with explicit type abstraction and type application. Types and terms use de Bruijn indices; the reduction relation contains both term- β and type- β steps and is closed under all term contexts. The proof combines Girard-style reducibility candidates [5] for an untyped lambda-calculus erasure with an explicit type-syntax measure that reflects the erased normalization result back to the full calculus. The final theorem states that every well-typed System F term is strongly normalizing, with the so the usual closed-term result is an immediate instance.

1 Overview

System F extends the simply typed lambda calculus with universal types, polymorphic abstraction, and type instantiation. The formalization represents these constructs explicitly:

$$\text{TmTAbs } M \quad \text{and} \quad \text{TmTApp } M A.$$

Typing uses de Bruijn indices for both term and type binders. The relation $\longrightarrow_\beta$ includes ordinary term- β , type- β , and compatible-context rules.

The semantic part interprets each type as a reducibility candidate in the untyped lambda calculus. Universal types quantify over all candidates, so polymorphic instantiation is handled semantically rather than by collapsing all type variables to one simple type. Erasure removes only type syntax and annotations. Every full reduction step either produces an erased untyped- β step or leaves the erasure unchanged while decreasing the number of explicit type abstractions and applications. A lexicographic reflection argument therefore yields strong normalization for the original System F term.

The proof proceeds in three stages. First, reducibility candidates establish strong normalization of the untyped erasure of every well-typed term. Second, each System F reduction is shown either to induce an untyped β -step after erasure or to preserve the erasure while strictly decreasing the amount of explicit type syntax. Finally, a lexicographic argument reflects strong normalization of the erasure back to the Church-style calculus.

Earlier formalizations include Altenkirch’s LEGO development [1], the ATS/LF development of Donnelly and Xi [4], and the Isabelle/HOL development of Popescu, Gunter, and Osborn [6]. The latter formalization uses Curry-style terms and HOAS on top of FOAS. The present entry instead treats Church-style syntax directly: term and type binders use de Bruijn indices, type- β is an explicit reduction rule, and the erasure proof reflects normalization with an explicit type-syntax measure. This distinguishes the calculus, binding representation, and proof architecture without claiming novelty for the classical normalization theorem.

We also considered basing the development on Berghofer’s **StrongNorm** formalization in the Isabelle/HOL **HOL-Proofs-Lambda** library [2]. That development provides a closely related de Bruijn treatment of strong normalization for the simply typed lambda calculus, including lifting, substitution, subject reduction, and an inductive characterization of terminating terms. It has only term abstraction/application, simple arrow types, and term- β , however; it does not cover polymorphic type abstraction/application, impredicative universal types, or type- β reduction. The present entry therefore follows its de Bruijn and substitution discipline while developing a System-F-specific reducibility-candidate semantics and an erasure/reflection argument for explicit type reductions. Berghofer’s **POPLmark AFP** entry [3] is similarly valuable for the de Bruijn treatment of binding and substitution, but it formalizes the type-safety infrastructure for System $F_{<}$, not strong normalization. We use these developments as methodological references rather than claiming to extend either source by direct import.

The development also proves subject reduction independently of the normalization argument. It is included both as a useful metatheorem and as an independent validation of the substitution infrastructure; the normalization proof does not invoke preservation. Typed weakening and term/type substitution lemmas validate the de Bruijn operations at both term- β and type- β redexes, and the preservation theorem provides an independent consistency check on the operational semantics.

```
theory System-F
  imports Main
begin
```

```
datatype dtty =
  TyVar nat
| TyArr dtty dtty
```

```

| TyAll dtv

datatype dtm =
  TmVar nat
| TmApp dtm dtm
| TmLam dtv dtm
| TmTAbs dtm
| TmTApp dtm dtv

fun ty-shift :: nat ⇒ dtv ⇒ dtv where
  ty-shift k (TyVar n) =
    (if n < k then TyVar n else TyVar (Suc n))
| ty-shift k (TyArr A B) = TyArr (ty-shift k A) (ty-shift k B)
| ty-shift k (TyAll A) = TyAll (ty-shift (Suc k) A)

fun ty-subst :: nat ⇒ dtv ⇒ dtv ⇒ dtv where
  ty-subst k S (TyVar n) =
    (if n < k then TyVar n else
     if n = k then S else TyVar (n - 1))
| ty-subst k S (TyArr A B) =
  TyArr (ty-subst k S A) (ty-subst k S B)
| ty-subst k S (TyAll A) =
  TyAll (ty-subst (Suc k) (ty-shift 0 S) A)

definition ty-subst0 :: dtv ⇒ dtv ⇒ dtv
where
  ty-subst0 S A = ty-subst 0 S A

fun tm-shift :: nat ⇒ dtm ⇒ dtm where
  tm-shift k (TmVar n) =
    (if n < k then TmVar n else TmVar (Suc n))
| tm-shift k (TmApp M N) = TmApp (tm-shift k M) (tm-shift k N)
| tm-shift k (TmLam A M) = TmLam A (tm-shift (Suc k) M)
| tm-shift k (TmTAbs M) = TmTAbs (tm-shift k M)
| tm-shift k (TmTApp M A) = TmTApp (tm-shift k M) A

fun tm-ty-shift :: nat ⇒ dtm ⇒ dtm where
  tm-ty-shift k (TmVar n) = TmVar n
| tm-ty-shift k (TmApp M P) =
  TmApp (tm-ty-shift k M) (tm-ty-shift k P)
| tm-ty-shift k (TmLam A M) =
  TmLam (ty-shift k A) (tm-ty-shift k M)
| tm-ty-shift k (TmTAbs M) =
  TmTAbs (tm-ty-shift (Suc k) M)
| tm-ty-shift k (TmTApp M A) =
  TmTApp (tm-ty-shift k M) (ty-shift k A)

fun tm-subst :: nat ⇒ dtm ⇒ dtm ⇒ dtm where
  tm-subst k N (TmVar n) =

```

```

      (if  $n < k$  then  $TmVar\ n$  else
       if  $n = k$  then  $N$  else  $TmVar\ (n - 1)$ )
|  $tm\text{-}subst\ k\ N\ (TmApp\ M\ P) =$ 
   $TmApp\ (tm\text{-}subst\ k\ N\ M)\ (tm\text{-}subst\ k\ N\ P)$ 
|  $tm\text{-}subst\ k\ N\ (TmLam\ A\ M) =$ 
   $TmLam\ A\ (tm\text{-}subst\ (Suc\ k)\ (tm\text{-}shift\ 0\ N)\ M)$ 
|  $tm\text{-}subst\ k\ N\ (TmTAbs\ M) =$ 
   $TmTAbs\ (tm\text{-}subst\ k\ (tm\text{-}ty\text{-}shift\ 0\ N)\ M)$ 
|  $tm\text{-}subst\ k\ N\ (TmTApp\ M\ A) =$ 
   $TmTApp\ (tm\text{-}subst\ k\ N\ M)\ A$ 

```

definition $tm\text{-}subst0 :: dtm \Rightarrow dtm \Rightarrow dtm$

where

$tm\text{-}subst0\ N\ M = tm\text{-}subst\ 0\ N\ M$

fun $tm\text{-}ty\text{-}subst :: nat \Rightarrow dty \Rightarrow dtm \Rightarrow dtm$ **where**

```

   $tm\text{-}ty\text{-}subst\ k\ S\ (TmVar\ n) = TmVar\ n$ 
|  $tm\text{-}ty\text{-}subst\ k\ S\ (TmApp\ M\ N) =$ 
   $TmApp\ (tm\text{-}ty\text{-}subst\ k\ S\ M)\ (tm\text{-}ty\text{-}subst\ k\ S\ N)$ 
|  $tm\text{-}ty\text{-}subst\ k\ S\ (TmLam\ A\ M) =$ 
   $TmLam\ (ty\text{-}subst\ k\ S\ A)\ (tm\text{-}ty\text{-}subst\ k\ S\ M)$ 
|  $tm\text{-}ty\text{-}subst\ k\ S\ (TmTAbs\ M) =$ 
   $TmTAbs\ (tm\text{-}ty\text{-}subst\ (Suc\ k)\ (ty\text{-}shift\ 0\ S)\ M)$ 
|  $tm\text{-}ty\text{-}subst\ k\ S\ (TmTApp\ M\ A) =$ 
   $TmTApp\ (tm\text{-}ty\text{-}subst\ k\ S\ M)\ (ty\text{-}subst\ k\ S\ A)$ 

```

definition $tm\text{-}ty\text{-}subst0 :: dty \Rightarrow dtm \Rightarrow dtm$

where

$tm\text{-}ty\text{-}subst0\ S\ M = tm\text{-}ty\text{-}subst\ 0\ S\ M$

definition $ty\text{-}env\text{-}cons :: 'a \Rightarrow (nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a$

where

$ty\text{-}env\text{-}cons\ C\ \varrho\ n = (case\ n\ of\ 0 \Rightarrow C \mid Suc\ k \Rightarrow \varrho\ k)$

definition $tm\text{-}env\text{-}cons :: dtm \Rightarrow (nat \Rightarrow dtm) \Rightarrow nat \Rightarrow dtm$

where

$tm\text{-}env\text{-}cons\ N\ \delta\ n = (case\ n\ of\ 0 \Rightarrow N \mid Suc\ k \Rightarrow \delta\ k)$

definition $tm\text{-}env\text{-}shift :: (nat \Rightarrow dtm) \Rightarrow nat \Rightarrow dtm$

where

$tm\text{-}env\text{-}shift\ \delta\ n =$
 $(case\ n\ of\ 0 \Rightarrow TmVar\ 0 \mid Suc\ k \Rightarrow tm\text{-}shift\ 0\ (\delta\ k))$

fun $tm\text{-}subst\text{-}env :: (nat \Rightarrow dtm) \Rightarrow dtm \Rightarrow dtm$ **where**

```

   $tm\text{-}subst\text{-}env\ \delta\ (TmVar\ k) = \delta\ k$ 
|  $tm\text{-}subst\text{-}env\ \delta\ (TmApp\ M\ N) =$ 
   $TmApp\ (tm\text{-}subst\text{-}env\ \delta\ M)\ (tm\text{-}subst\text{-}env\ \delta\ N)$ 
|  $tm\text{-}subst\text{-}env\ \delta\ (TmLam\ A\ M) =$ 
   $TmLam\ A\ (tm\text{-}subst\text{-}env\ (tm\text{-}env\text{-}shift\ \delta)\ M)$ 

```

| $tm\text{-}subst\text{-}env\ \delta\ (TmTAbs\ M) = TmTAbs\ (tm\text{-}subst\text{-}env\ \delta\ M)$
| $tm\text{-}subst\text{-}env\ \delta\ (TmTApp\ M\ A) =$
 $\quad TmTApp\ (tm\text{-}subst\text{-}env\ \delta\ M)\ A$

inductive $wf\text{-}ty :: nat \Rightarrow dty \Rightarrow bool$ **where**
 $wf\text{-}var: n < k \implies wf\text{-}ty\ k\ (TyVar\ n)$
| $wf\text{-}arr: \llbracket wf\text{-}ty\ k\ A; wf\text{-}ty\ k\ B \rrbracket \implies$
 $\quad wf\text{-}ty\ k\ (TyArr\ A\ B)$
| $wf\text{-}all: wf\text{-}ty\ (Suc\ k)\ A \implies wf\text{-}ty\ k\ (TyAll\ A)$

inductive $ctx\text{-}mem :: dty\ list \Rightarrow nat \Rightarrow dty \Rightarrow bool$ **where**
 $ctx\text{-}zero: ctx\text{-}mem\ (A \# \Gamma)\ 0\ A$
| $ctx\text{-}suc: ctx\text{-}mem\ \Gamma\ k\ A \implies ctx\text{-}mem\ (B \# \Gamma)\ (Suc\ k)\ A$

inductive $typing :: nat \Rightarrow dty\ list \Rightarrow dtm \Rightarrow dty \Rightarrow bool$
 $(\langle - \rangle; - \vdash - : \rightarrow [60,60,60,60]\ 60)$ **where**
 $ty\text{-}var: ctx\text{-}mem\ \Gamma\ n\ A \implies typing\ k\ \Gamma\ (TmVar\ n)\ A$
| $ty\text{-}app: \llbracket typing\ k\ \Gamma\ M\ (TyArr\ A\ B); typing\ k\ \Gamma\ N\ A \rrbracket$
 $\implies typing\ k\ \Gamma\ (TmApp\ M\ N)\ B$
| $ty\text{-}lam: \llbracket wf\text{-}ty\ k\ A; typing\ k\ (A \# \Gamma)\ M\ B \rrbracket$
 $\implies typing\ k\ \Gamma\ (TmLam\ A\ M)\ (TyArr\ A\ B)$
| $ty\text{-}tabs: typing\ (Suc\ k)\ (map\ (ty\text{-}shift\ 0)\ \Gamma)\ M\ B$
 $\implies typing\ k\ \Gamma\ (TmTAbs\ M)\ (TyAll\ B)$
| $ty\text{-}tapp: \llbracket typing\ k\ \Gamma\ M\ (TyAll\ B); wf\text{-}ty\ k\ A \rrbracket$
 $\implies typing\ k\ \Gamma\ (TmTApp\ M\ A)\ (ty\text{-}subst0\ A\ B)$

inductive $beta :: dtm \Rightarrow dtm \Rightarrow bool$
 $(\langle - \rangle \rightarrow_\beta \rightarrow [80,80]\ 80)$ **where**
 $beta\text{-}appL: M \rightarrow_\beta M' \implies$
 $\quad TmApp\ M\ N \rightarrow_\beta TmApp\ M'\ N$
| $beta\text{-}appR: N \rightarrow_\beta N' \implies$
 $\quad TmApp\ M\ N \rightarrow_\beta TmApp\ M\ N'$
| $beta\text{-}lam: M \rightarrow_\beta M' \implies$
 $\quad TmLam\ A\ M \rightarrow_\beta TmLam\ A\ M'$
| $beta\text{-}tabs: M \rightarrow_\beta M' \implies$
 $\quad TmTAbs\ M \rightarrow_\beta TmTAbs\ M'$
| $beta\text{-}tapp: M \rightarrow_\beta M' \implies$
 $\quad TmTApp\ M\ A \rightarrow_\beta TmTApp\ M'\ A$
| $beta\text{-}term: TmApp\ (TmLam\ A\ M)\ N \rightarrow_\beta tm\text{-}subst0\ N\ M$
| $beta\text{-}type: TmTApp\ (TmTAbs\ M)\ A \rightarrow_\beta tm\text{-}ty\text{-}subst0\ A\ M$

inductive $SN :: dtm \Rightarrow bool$ **where**
 $SN\text{-}intro: (\bigwedge M'. M \rightarrow_\beta M' \implies SN\ M') \implies SN\ M$

end
theory *System-F-Subject-Reduction*
imports *System-F*
begin

This theory records the independent type-preservation check for the de

Bruijn presentation. It is not needed by the normalization argument, but it validates the substitution operations used by both beta rules.

fun *ctx-insert* :: *nat* $\Rightarrow$ *dtty* $\Rightarrow$ *dtty list* $\Rightarrow$ *dtty list* **where**
 ctx-insert *i* *B* [] = [*B*]
 | *ctx-insert* 0 *B* (*A* # Γ) = *B* # *A* # Γ
 | *ctx-insert* (*Suc i*) *B* (*A* # Γ) = *A* # *ctx-insert i B* Γ

lemma *ctx-insert-Suc-cons*:
 ctx-insert (*Suc i*) *B* (*A* # Γ) = *A* # *ctx-insert i B* Γ
 <proof>

lemma *map-ctx-insert*:
 map *f* (*ctx-insert i B* Γ) = *ctx-insert i* (*f B*) (*map f* Γ)
 <proof>

lemma *ctx-mem-ctx-insert*:
 assumes *ctx-mem* Γ *n* *A*
 shows *ctx-mem* (*ctx-insert i B* Γ)
 (*if* *n* < *i* *then n* *else Suc n*) *A*
 <proof>

lemma *ctx-mem-sucD*:
 ctx-mem (*B* # Γ) (*Suc n*) *A* $\implies$ *ctx-mem* Γ *n* *A*
 <proof>

lemma *ctx-mem-insert-cases*:
 assumes *ctx-mem* (*ctx-insert i C* Γ) *n* *A* *i* $\leq$ *length* Γ
 shows (*n* < *i* $\wedge$ *ctx-mem* Γ *n* *A*) $\vee$
 (*n* = *i* $\wedge$ *A* = *C*) $\vee$
 (*i* < *n* $\wedge$ *ctx-mem* Γ (*n* - 1) *A*)
 <proof>

lemma *typing-weakening*:
 typing *k* Γ *M* *A* $\implies$
 typing *k* (*ctx-insert i B* Γ) (*tm-shift i M*) *A*
 <proof>

lemma *wf-ty-shift-at*:
 assumes *wf-ty* *k* *A* *l* $\leq$ *k*
 shows *wf-ty* (*Suc k*) (*ty-shift l A*)
 <proof>

lemma *wf-ty-shift*:
 assumes *wf-ty* *k* *A*
 shows *wf-ty* (*Suc k*) (*ty-shift 0 A*)
 <proof>

lemma *ctx-mem-map*:
 assumes *ctx-mem* Γ *n* *A*

shows $ctx\text{-}mem \ (map \ f \ \Gamma) \ n \ (f \ A)$
 $\langle proof \rangle$

lemma *ty-shift-commute*:
assumes $i \leq j$
shows $ty\text{-}shift \ i \ (ty\text{-}shift \ j \ A) =$
 $ty\text{-}shift \ (Suc \ j) \ (ty\text{-}shift \ i \ A)$
 $\langle proof \rangle$

lemma *ty-subst-shift*:
assumes $l \leq k$
shows $ty\text{-}subst \ (Suc \ k) \ (ty\text{-}shift \ l \ S) \ (ty\text{-}shift \ l \ A) =$
 $ty\text{-}shift \ l \ (ty\text{-}subst \ k \ S \ A)$
 $\langle proof \rangle$

lemma *ty-subst-ty-shift*:
 $ty\text{-}subst \ (Suc \ k) \ (ty\text{-}shift \ 0 \ S) \ (ty\text{-}shift \ 0 \ A) =$
 $ty\text{-}shift \ 0 \ (ty\text{-}subst \ k \ S \ A)$
 $\langle proof \rangle$

lemma *ty-subst-shift-suc*:
assumes $k \leq l$
shows $ty\text{-}subst \ k \ (ty\text{-}shift \ l \ S) \ (ty\text{-}shift \ (Suc \ l) \ A) =$
 $ty\text{-}shift \ l \ (ty\text{-}subst \ k \ S \ A)$
 $\langle proof \rangle$

lemma *typing-ty-shift-at*:
assumes $typing \ k \ \Gamma \ M \ A \ l \leq k$
shows $typing \ (Suc \ k) \ (map \ (ty\text{-}shift \ l) \ \Gamma)$
 $(tm\text{-}ty\text{-}shift \ l \ M) \ (ty\text{-}shift \ l \ A)$
 $\langle proof \rangle$

lemma *typing-ty-shift*:
 $typing \ k \ \Gamma \ M \ A \implies$
 $typing \ (Suc \ k) \ (map \ (ty\text{-}shift \ 0) \ \Gamma)$
 $(tm\text{-}ty\text{-}shift \ 0 \ M) \ (ty\text{-}shift \ 0 \ A)$
 $\langle proof \rangle$

lemma *wf-ty-subst-pred*:
assumes $wf\text{-}ty \ n \ A \ wf\text{-}ty \ (n - 1) \ S \ j < n$
shows $wf\text{-}ty \ (n - 1) \ (ty\text{-}subst \ j \ S \ A)$
 $\langle proof \rangle$

lemma *wf-ty-subst-at*:
assumes $wf\text{-}ty \ (Suc \ k) \ A \ wf\text{-}ty \ k \ S \ j \leq k$
shows $wf\text{-}ty \ k \ (ty\text{-}subst \ j \ S \ A)$
 $\langle proof \rangle$

lemma *ty-subst-shift-id*:

$ty\text{-subst } k \ T \ (ty\text{-shift } k \ A) = A$
 $\langle proof \rangle$

lemma *ty-subst-comp-at*:

$i \leq j \longrightarrow$
 $ty\text{-subst } i \ (ty\text{-subst } j \ S \ T)$
 $(ty\text{-subst } (Suc \ j) \ (ty\text{-shift } i \ S) \ A) =$
 $ty\text{-subst } j \ S \ (ty\text{-subst } i \ T \ A)$
 $\langle proof \rangle$

lemma *ty-subst-comp*:

$ty\text{-subst } j \ S \ (ty\text{-subst } 0 \ T \ A) =$
 $ty\text{-subst } 0 \ (ty\text{-subst } j \ S \ T)$
 $(ty\text{-subst } (Suc \ j) \ (ty\text{-shift } 0 \ S) \ A)$
 $\langle proof \rangle$

lemma *typing-ty-subst-at-aux*:

assumes $typing \ n \ \Gamma \ M \ A \ n = Suc \ k \ wf\text{-ty} \ k \ S \ j \leq k$
shows $typing \ k \ (\text{map } (ty\text{-subst } j \ S) \ \Gamma)$
 $(tm\text{-ty-subst } j \ S \ M) \ (ty\text{-subst } j \ S \ A)$
 $\langle proof \rangle$

lemma *typing-ty-subst-at*:

assumes $typing \ (Suc \ k) \ \Gamma \ M \ A \ wf\text{-ty} \ k \ S \ j \leq k$
shows $typing \ k \ (\text{map } (ty\text{-subst } j \ S) \ \Gamma)$
 $(tm\text{-ty-subst } j \ S \ M) \ (ty\text{-subst } j \ S \ A)$
 $\langle proof \rangle$

lemma *typing-ty-subst*:

assumes $typing \ (Suc \ k) \ \Gamma \ M \ A \ wf\text{-ty} \ k \ S$
shows $typing \ k \ (\text{map } (ty\text{-subst } 0 \ S) \ \Gamma)$
 $(tm\text{-ty-subst } 0 \ S \ M) \ (ty\text{-subst } 0 \ S \ A)$
 $\langle proof \rangle$

lemma *typing-tm-subst-at*:

assumes $typing \ k \ \Gamma \ N \ C$
 $typing \ k \ (ctx\text{-insert } i \ C \ \Gamma) \ M \ A$
 $i \leq \text{length } \Gamma$
shows $typing \ k \ \Gamma \ (tm\text{-subst } i \ N \ M) \ A$
 $\langle proof \rangle$

lemma *typing-tm-subst0*:

assumes $typing \ k \ \Gamma \ N \ C$
 $typing \ k \ (C \ \# \ \Gamma) \ M \ A$
shows $typing \ k \ \Gamma \ (tm\text{-subst0 } N \ M) \ A$
 $\langle proof \rangle$

theorem *subject-reduction*:

assumes $typing \ k \ \Gamma \ M \ A \ M \longrightarrow_{\beta} M'$

```

shows typing  $k \ \Gamma \ M' \ A$ 
   $\langle \text{proof} \rangle$ 

end
theory System-F-Untyped
  imports System-F
begin

datatype ulam =
  | UVar nat
  | UApp ulam ulam
  | ULam ulam

fun u-shift :: nat  $\Rightarrow$  ulam  $\Rightarrow$  ulam where
  u-shift  $k$  (UVar  $n$ ) =
    (if  $n < k$  then UVar  $n$  else UVar (Suc  $n$ ))
  | u-shift  $k$  (UApp  $M$   $N$ ) = UApp (u-shift  $k$   $M$ ) (u-shift  $k$   $N$ )
  | u-shift  $k$  (ULam  $M$ ) = ULam (u-shift (Suc  $k$ )  $M$ )

fun u-subst :: nat  $\Rightarrow$  ulam  $\Rightarrow$  ulam  $\Rightarrow$  ulam where
  u-subst  $k$   $N$  (UVar  $n$ ) =
    (if  $n < k$  then UVar  $n$  else
     if  $n = k$  then  $N$  else UVar ( $n - 1$ ))
  | u-subst  $k$   $N$  (UApp  $M$   $P$ ) =
    UApp (u-subst  $k$   $N$   $M$ ) (u-subst  $k$   $N$   $P$ )
  | u-subst  $k$   $N$  (ULam  $M$ ) =
    ULam (u-subst (Suc  $k$ ) (u-shift 0  $N$ )  $M$ )

definition u-subst0 :: ulam  $\Rightarrow$  ulam  $\Rightarrow$  ulam
where u-subst0  $N$   $M$  = u-subst 0  $N$   $M$ 

lemma u-subst-shift:
  u-subst  $k$   $N$  (u-shift  $k$   $M$ ) =  $M$ 
   $\langle \text{proof} \rangle$ 

lemma u-shift-shift:
   $l \leq k \longrightarrow$ 
  u-shift  $l$  (u-shift  $k$   $M$ ) =
  u-shift (Suc  $k$ ) (u-shift  $l$   $M$ )
   $\langle \text{proof} \rangle$ 

lemma u-shift-subst-comm:
   $l \leq k \longrightarrow$ 
  u-shift  $l$  (u-subst  $k$   $N$   $M$ ) =
  u-subst (Suc  $k$ ) (u-shift  $l$   $N$ ) (u-shift  $l$   $M$ )
   $\langle \text{proof} \rangle$ 

lemma u-subst-comp-at:
   $l \leq k \longrightarrow$ 

```

$u\text{-subst } l \ (u\text{-subst } k \ N \ P)$
 $(u\text{-subst } (Suc \ k) \ (u\text{-shift } l \ N) \ M) =$
 $u\text{-subst } k \ N \ (u\text{-subst } l \ P \ M)$
 $\langle proof \rangle$

inductive $ubeta :: ulam \Rightarrow ulam \Rightarrow bool$
 $(\langle - \longrightarrow_u - \rangle [80,80] \ 80)$ **where**
 $uappL: M \longrightarrow_u M' \Longrightarrow$
 $UApp \ M \ N \longrightarrow_u UApp \ M' \ N$
 $| \ uappR: N \longrightarrow_u N' \Longrightarrow$
 $UApp \ M \ N \longrightarrow_u UApp \ M \ N'$
 $| \ ulam: M \longrightarrow_u M' \Longrightarrow$
 $ULam \ M \longrightarrow_u ULam \ M'$
 $| \ ured: UApp \ (ULam \ M) \ N \longrightarrow_u u\text{-subst0 } N \ M$

inductive $uSN :: ulam \Rightarrow bool$ **where**
 $uSN\text{-intro}: (\bigwedge M'. M \longrightarrow_u M' \Longrightarrow uSN \ M') \Longrightarrow uSN \ M$

lemma $uSN\text{-preserved}$:
assumes $uSN \ M$ **and** $M \longrightarrow_u M'$
shows $uSN \ M'$
 $\langle proof \rangle$

lemma $u\text{-subst-beta-at}$:
assumes $M \longrightarrow_u M'$
shows $u\text{-subst } k \ N \ M \longrightarrow_u u\text{-subst } k \ N \ M'$
 $\langle proof \rangle$

lemma $uSN\text{-subst-beta}$:
assumes $M \longrightarrow_u M'$
shows $u\text{-subst0 } N \ M \longrightarrow_u u\text{-subst0 } N \ M'$
 $\langle proof \rangle$

lemma $uSN\text{-reflect-subst}$:
assumes $uSN \ (u\text{-subst0 } (UVar \ 0) \ M)$
shows $uSN \ M$
 $\langle proof \rangle$

inductive $uFst :: ulam \Rightarrow ulam \Rightarrow bool$ **where**
 $uFst\text{-intro}: uFst \ (UApp \ M \ N) \ M$

lemma $uSN\text{-of-fst}$:
assumes $uSN \ (UApp \ M \ N)$
shows $uSN \ M$
 $\langle proof \rangle$

lemma $u\text{double-}SN\text{-aux}$:
assumes $a: uSN \ a$
and $b: uSN \ b$

and *hyp*: $\bigwedge x z.$
 $\llbracket \bigwedge y. x \rightarrow_u y \implies uSN\ y;$
 $\bigwedge y. x \rightarrow_u y \implies P\ y\ z;$
 $\bigwedge u. z \rightarrow_u u \implies uSN\ u;$
 $\bigwedge u. z \rightarrow_u u \implies P\ x\ u \rrbracket \implies P\ x\ z$
shows $P\ a\ b$
 $\langle proof \rangle$

lemma *udouble-SN*[*consumes 2*]:
assumes $a: uSN\ a$
and $b: uSN\ b$
and $c: \bigwedge x z.$
 $\llbracket \bigwedge y. x \rightarrow_u y \implies P\ y\ z;$
 $\bigwedge u. z \rightarrow_u u \implies P\ x\ u \rrbracket \implies P\ x\ z$
shows $P\ a\ b$
 $\langle proof \rangle$

definition *uneutral* :: *ulam set* $\Rightarrow$ *bool* **where**
 $uneutral\ M \longleftrightarrow$
 $(\exists k. M = UVar\ k) \vee$
 $(\exists P\ Q. M = UApp\ P\ Q)$

The neutral terms used by the candidate condition are variables and arbitrary applications. In particular, applications are considered neutral even when their head is an abstraction. This deliberately broad formulation makes the candidate closure stable under the full compatible reduction relation; the redex case is handled explicitly when the saturation lemmas analyze the reducts of an application.

lemma *uneutral-var*:
 $uneutral\ (UVar\ k)$
 $\langle proof \rangle$

lemma *uneutral-app*:
 $uneutral\ (UApp\ M\ N)$
 $\langle proof \rangle$

definition *uCR1* :: *ulam set* $\Rightarrow$ *bool* **where**
 $uCR1\ C \longleftrightarrow (\forall M. M \in C \longrightarrow uSN\ M)$

definition *uCR2* :: *ulam set* $\Rightarrow$ *bool* **where**
 $uCR2\ C \longleftrightarrow$
 $(\forall M\ M'. M \in C \longrightarrow M \rightarrow_u M' \longrightarrow M' \in C)$

definition *uCR3* :: *ulam set* $\Rightarrow$ *bool* **where**
 $uCR3\ C \longleftrightarrow$
 $(\forall M. uneutral\ M \longrightarrow$
 $(\forall M'. M \rightarrow_u M' \longrightarrow M' \in C) \longrightarrow M \in C)$

definition *ucandidate* :: *ulam set* $\Rightarrow$ *bool* **where**

$$ucandidate\ C \longleftrightarrow uCR1\ C \wedge uCR2\ C \wedge uCR3\ C$$

definition $uSN\text{-}set :: ulam\ set$ **where**

$$uSN\text{-}set = \{M. uSN\ M\}$$

lemma $uSN\text{-}set\text{-}candidate$:

$$ucandidate\ uSN\text{-}set$$

$\langle proof \rangle$

lemma $uvar\text{-}mem\text{-}candidate$:

$$\text{assumes } ucandidate\ C$$

$$\text{shows } UVar\ k \in C$$

$\langle proof \rangle$

definition $uarr :: ulam\ set \Rightarrow ulam\ set \Rightarrow ulam\ set$ **where**

$$uarr\ C\ D = \{M. \forall N. N \in C \longrightarrow UApp\ M\ N \in D\}$$

lemma $uarr\text{-}CR2$:

$$\text{assumes } uCR2\ D$$

$$\text{shows } uCR2\ (uarr\ C\ D)$$

$\langle proof \rangle$

lemma $uarr\text{-}CR1$:

$$\text{assumes } uCR1\ C \text{ and } uCR1\ D \text{ and } uCR3\ C$$

$$\text{shows } uCR1\ (uarr\ C\ D)$$

$\langle proof \rangle$

lemma $uarr\text{-}CR3$:

$$\text{assumes } uCR1\ C \text{ and } uCR2\ C \text{ and } uCR3\ C$$

$$\text{and } uCR3\ D$$

$$\text{shows } uCR3\ (uarr\ C\ D)$$

$\langle proof \rangle$

lemma $uarr\text{-}candidate$:

$$\text{assumes } ucandidate\ C \text{ and } ucandidate\ D$$

$$\text{shows } ucandidate\ (uarr\ C\ D)$$

$\langle proof \rangle$

lemma $ubeta\text{-}lam\text{-}cases$:

$$\text{assumes } ULam\ M \longrightarrow_u P$$

$$\text{obtains } M' \text{ where } M \longrightarrow_u M' \text{ and } P = ULam\ M'$$

$\langle proof \rangle$

The abstraction lemma is the semantic heart of the untyped development. Its premise says that substituting every argument from the domain candidate into the body produces an element of the codomain candidate. The proof first obtains strong normalization of the body by testing it with a fresh variable, and then uses simultaneous induction on body and argument reductions to establish the candidate condition for the application of the

abstraction.

lemma *uabs-RED*:

assumes C : *ucandidate* C
and D : *ucandidate* D
and asm : $\forall s. s \in C \longrightarrow u\text{-subst0 } s \ M \in D$
shows $ULam \ M \in uarr \ C \ D$

$\langle proof \rangle$

end

theory *System-F-Erasure*

imports *System-F System-F-Untyped*

begin

fun *f-erase* :: $dtm \Rightarrow ulam$ **where**

$f\text{-erase } (TmVar \ n) = UVar \ n$
 $| f\text{-erase } (TmApp \ M \ N) = UApp \ (f\text{-erase } M) \ (f\text{-erase } N)$
 $| f\text{-erase } (TmLam \ A \ M) = ULam \ (f\text{-erase } M)$
 $| f\text{-erase } (TmTAbs \ M) = f\text{-erase } M$
 $| f\text{-erase } (TmTApp \ M \ A) = f\text{-erase } M$

lemma *f-erase-tm-shift*:

$f\text{-erase } (tm\text{-shift } k \ M) = u\text{-shift } k \ (f\text{-erase } M)$
 $\langle proof \rangle$

lemma *f-erase-tm-ty-shift*:

$f\text{-erase } (tm\text{-ty-shift } k \ M) = f\text{-erase } M$
 $\langle proof \rangle$

lemma *f-erase-tm-subst*:

$f\text{-erase } (tm\text{-subst } k \ N \ M) =$
 $u\text{-subst } k \ (f\text{-erase } N) \ (f\text{-erase } M)$
 $\langle proof \rangle$

lemma *f-erase-tm-ty-subst*:

$f\text{-erase } (tm\text{-ty-subst } k \ S \ M) = f\text{-erase } M$
 $\langle proof \rangle$

lemma *f-erase-tm-subst0*:

$f\text{-erase } (tm\text{-subst0 } N \ M) =$
 $u\text{-subst0 } (f\text{-erase } N) \ (f\text{-erase } M)$
 $\langle proof \rangle$

lemma *f-erase-tm-ty-subst0*:

$f\text{-erase } (tm\text{-ty-subst0 } S \ M) = f\text{-erase } M$
 $\langle proof \rangle$

fun *f-type-count* :: $dtm \Rightarrow nat$ **where**

$f\text{-type-count } (TmVar \ n) = 0$
 $| f\text{-type-count } (TmApp \ M \ N) = f\text{-type-count } M + f\text{-type-count } N$

$| f\text{-type-count } (TmLam\ A\ M) = f\text{-type-count } M$
 $| f\text{-type-count } (TmTAbs\ M) = Suc\ (f\text{-type-count } M)$
 $| f\text{-type-count } (TmTApp\ M\ A) = Suc\ (f\text{-type-count } M)$

lemma *f-type-count-tm-ty-subst*:
 $f\text{-type-count } (tm\text{-ty-subst } k\ S\ M) = f\text{-type-count } M$
 $\langle proof \rangle$

lemma *f-type-count-tm-ty-subst0*:
 $f\text{-type-count } (tm\text{-ty-subst0 } S\ M) = f\text{-type-count } M$
 $\langle proof \rangle$

lemma *f-beta-progress*:
assumes $M \longrightarrow_{\beta} M'$
shows $ubeta\ (f\text{-erase } M)\ (f\text{-erase } M') \vee$
 $f\text{-erase } M = f\text{-erase } M' \wedge$
 $f\text{-type-count } M' < f\text{-type-count } M$
 $\langle proof \rangle$

lemma *fSN-reflect*:
assumes $uSN\ (f\text{-erase } M)$
shows $SN\ M$
 $\langle proof \rangle$

end
theory *System-F-Normalization*
imports *System-F-Erasure*
begin

definition *f-env-ok* :: $(nat \Rightarrow ulam\ set) \Rightarrow bool$ **where**
 $f\text{-env-ok } \varrho \longleftrightarrow (\forall k. u\text{candidate } (\varrho\ k))$

definition *f-all-sem* :: $(ulam\ set \Rightarrow ulam\ set) \Rightarrow ulam\ set$ **where**
 $f\text{-all-sem } F = \{M. \forall C. u\text{candidate } C \longrightarrow M \in F\ C\}$

lemma *f-all-candidate*:
assumes $H: \forall C. u\text{candidate } C \longrightarrow u\text{candidate } (F\ C)$
shows $u\text{candidate } (f\text{-all-sem } F)$
 $\langle proof \rangle$

definition *f-env-insert* ::
 $nat \Rightarrow ulam\ set \Rightarrow (nat \Rightarrow ulam\ set) \Rightarrow$
 $nat \Rightarrow ulam\ set$
where
 $f\text{-env-insert } k\ C\ \varrho\ n =$
 $(if\ n < k\ then\ \varrho\ n\ else$
 $if\ n = k\ then\ C\ else\ \varrho\ (n - 1))$

lemma *f-env-insert-zero*:

$f\text{-env-insert } 0 \ C \ \varrho = ty\text{-env-cons } C \ \varrho$
 $\langle proof \rangle$

lemma $f\text{-env-insert-suc-cons}$:
 $f\text{-env-insert } (Suc \ k) \ C \ (ty\text{-env-cons } D \ \varrho) =$
 $ty\text{-env-cons } D \ (f\text{-env-insert } k \ C \ \varrho)$
 $\langle proof \rangle$

lemma $f\text{-env-ok-cons}$:
assumes $f\text{-env-ok } \varrho$ **and** $ucandidate \ C$
shows $f\text{-env-ok } (ty\text{-env-cons } C \ \varrho)$
 $\langle proof \rangle$

fun $f\text{-sem} :: dt\vartheta \Rightarrow (nat \Rightarrow ulam \ set) \Rightarrow ulam \ set$ **where**
 $f\text{-sem } (TyVar \ k) \ \varrho = \varrho \ k$
 $| f\text{-sem } (TyArr \ A \ B) \ \varrho =$
 $uarr \ (f\text{-sem } A \ \varrho) \ (f\text{-sem } B \ \varrho)$
 $| f\text{-sem } (TyAll \ A) \ \varrho =$
 $f\text{-all-sem } (\lambda C. f\text{-sem } A \ (ty\text{-env-cons } C \ \varrho))$

lemma $f\text{-sem-candidate}$:
 $f\text{-env-ok } \varrho \longrightarrow ucandidate \ (f\text{-sem } A \ \varrho)$
 $\langle proof \rangle$

lemma $f\text{-sem-ty-shift}$:
 $f\text{-sem } (ty\text{-shift } k \ A) \ (f\text{-env-insert } k \ C \ \varrho) =$
 $f\text{-sem } A \ \varrho$
 $\langle proof \rangle$

lemma $f\text{-sem-ty-subst}$:
 $f\text{-sem } (ty\text{-subst } k \ S \ A) \ \varrho =$
 $f\text{-sem } A \ (f\text{-env-insert } k \ (f\text{-sem } S \ \varrho) \ \varrho)$
 $\langle proof \rangle$

definition $f\text{-tenv-cons} :: ulam \Rightarrow (nat \Rightarrow ulam) \Rightarrow$
 $nat \Rightarrow ulam$

where
 $f\text{-tenv-cons } N \ \eta \ n =$
 $(case \ n \ of \ 0 \Rightarrow N \ | \ Suc \ k \Rightarrow \eta \ k)$

definition $f\text{-tenv-shift} :: (nat \Rightarrow ulam) \Rightarrow nat \Rightarrow ulam$
where

$f\text{-tenv-shift } \eta \ n =$
 $(case \ n \ of \ 0 \Rightarrow UVar \ 0 \ | \ Suc \ k \Rightarrow u\text{-shift } 0 \ (\eta \ k))$

definition $f\text{-env-subst} ::$
 $nat \Rightarrow ulam \Rightarrow (nat \Rightarrow ulam) \Rightarrow$
 $nat \Rightarrow ulam$

where

$$f\text{-env-subst } k \ N \ \eta \ n = u\text{-subst } k \ N \ (\eta \ n)$$

lemma *f-env-subst-shift*:

$$\begin{aligned} f\text{-env-subst } (Suc \ k) \ (u\text{-shift } 0 \ N) \ (f\text{-tenv-shift } \eta) = \\ f\text{-tenv-shift } (f\text{-env-subst } k \ N \ \eta) \\ \langle proof \rangle \end{aligned}$$

lemma *f-env-subst-shift0*:

$$\begin{aligned} f\text{-env-subst } 0 \ N \ (f\text{-tenv-shift } \eta) = \\ f\text{-tenv-cons } N \ \eta \\ \langle proof \rangle \end{aligned}$$

fun *f-interp* :: (nat $\Rightarrow$ ulam) $\Rightarrow$ dtm $\Rightarrow$ ulam **where**

$$\begin{aligned} f\text{-interp } \eta \ (TmVar \ n) &= \eta \ n \\ | f\text{-interp } \eta \ (TmApp \ M \ N) &= \\ &\quad UApp \ (f\text{-interp } \eta \ M) \ (f\text{-interp } \eta \ N) \\ | f\text{-interp } \eta \ (TmLam \ A \ M) &= \\ &\quad ULam \ (f\text{-interp } (f\text{-tenv-shift } \eta) \ M) \\ | f\text{-interp } \eta \ (TmTAbs \ M) &= f\text{-interp } \eta \ M \\ | f\text{-interp } \eta \ (TmTApp \ M \ A) &= f\text{-interp } \eta \ M \end{aligned}$$

lemma *f-interp-subst*:

$$\begin{aligned} u\text{-subst } k \ N \ (f\text{-interp } \eta \ M) = \\ f\text{-interp } (f\text{-env-subst } k \ N \ \eta) \ M \\ \langle proof \rangle \end{aligned}$$

lemma *f-tenv-shift-id*:

$$\begin{aligned} f\text{-tenv-shift } (\lambda n. \ UVar \ n) = (\lambda n. \ UVar \ n) \\ \langle proof \rangle \end{aligned}$$

lemma *f-interp-id*:

$$\begin{aligned} f\text{-interp } (\lambda n. \ UVar \ n) \ M = f\text{-erase } M \\ \langle proof \rangle \end{aligned}$$

lemma *f-interp-subst0-shift*:

$$\begin{aligned} u\text{-subst0 } N \ (f\text{-interp } (f\text{-tenv-shift } \eta) \ M) = \\ f\text{-interp } (f\text{-tenv-cons } N \ \eta) \ M \\ \langle proof \rangle \end{aligned}$$

definition *f-tenv-ok* ::

$$\begin{aligned} dty \ list \Rightarrow (nat \Rightarrow ulam) \Rightarrow \\ (nat \Rightarrow ulam \ set) \Rightarrow bool \end{aligned}$$

where

$$\begin{aligned} f\text{-tenv-ok } \Gamma \ \eta \ \varrho \iff \\ (\forall n \ A. \ ctx\text{-mem } \Gamma \ n \ A \longrightarrow \\ \eta \ n \in f\text{-sem } A \ \varrho) \end{aligned}$$

lemma *f-ctx-mem-mapE*:

$$ctx\text{-mem } (map \ (ty\text{-shift } 0) \ \Gamma) \ n \ B \longrightarrow$$

($\exists A. \text{ctx-mem } \Gamma \ n \ A \wedge B = \text{ty-shift } 0 \ A$)
 $\langle \text{proof} \rangle$

lemma *f-tenv-ok-cons*:
 assumes *hctx*: *f-tenv-ok* $\Gamma \ \eta \ \varrho$
 and *hval*: $N \in \text{f-sem } A \ \varrho$
 shows *f-tenv-ok* $(A \ \# \ \Gamma) \ (\text{f-tenv-cons } N \ \eta) \ \varrho$
 $\langle \text{proof} \rangle$

lemma *f-tenv-ok-ty-shift*:
 assumes *hctx*: *f-tenv-ok* $\Gamma \ \eta \ \varrho$
 and *henv*: *f-env-ok* ϱ
 and *hcan*: *ucandidate* C
 shows *f-tenv-ok* $(\text{map } (\text{ty-shift } 0) \ \Gamma) \ \eta$
 $(\text{ty-env-cons } C \ \varrho)$
 $\langle \text{proof} \rangle$

definition *f-prop* ::
 $\text{dty list} \Rightarrow \text{dtm} \Rightarrow \text{dty} \Rightarrow \text{bool}$
where
 $\text{f-prop } \Gamma \ M \ A \longleftrightarrow$
 $(\forall \varrho \ \eta.$
 $\text{f-env-ok } \varrho \longrightarrow$
 $\text{f-tenv-ok } \Gamma \ \eta \ \varrho \longrightarrow$
 $\text{f-interp } \eta \ M \in \text{f-sem } A \ \varrho)$

lemma *f-fundamental*:
 $\text{typing } k \ \Gamma \ M \ A \implies \text{f-prop } \Gamma \ M \ A$
 $\langle \text{proof} \rangle$

lemma *f-identity-tenv-ok*:
 $\text{f-tenv-ok } \Gamma \ (\lambda n. \ \text{UVar } n) \ (\lambda -. \ \text{uSN-set})$
 $\langle \text{proof} \rangle$

lemma *f-welltyped-fundamental*:
 assumes *typing* $k \ \Gamma \ M \ A$
 shows *uSN* $(\text{f-erase } M)$
 $\langle \text{proof} \rangle$

theorem *strong-normalization*:
 assumes *typing* $k \ \Gamma \ M \ A$
 shows *SN* M
 $\langle \text{proof} \rangle$

end

References

- [1] T. Altenkirch. A formalization of the strong normalization proof for system f in lego. In *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 1993. DOI: <https://doi.org/10.1007/BFB0037095>.
- [2] S. Berghofer. Strong normalization for simply-typed lambda calculus. Isabelle/HOL library theory HOL/Proofs/Lambda/StrongNorm.thy, 2000. <https://isabelle.in.tum.de/library/HOL/HOL-Proofs-Lambda/document.pdf>.
- [3] S. Berghofer. Poplmark challenge via de bruijn indices. *Archive of Formal Proofs*, August 2007. <https://isa-afp.org/entries/POPLmark-deBruijn.html>, Formal proof development.
- [4] K. Donnelly and H. Xi. A formalization of strong normalization for simply-typed lambda-calculus and system f. *Electronic Notes in Theoretical Computer Science*, 174(5):109–125, 2007. DOI: <https://doi.org/10.1016/j.entcs.2007.01.021>.
- [5] J.-Y. Girard. Interprétation fonctionnelle et élimination des coupures de l’arithmétique d’ordre supérieur. *Thèse de doctorat, Université Paris VII*, 1972.
- [6] A. Popescu, E. L. Gunter, and C. J. Osborn. Strong normalization for system f by hoas on top of foas. In *2010 25th Annual IEEE Symposium on Logic in Computer Science*, pages 31–40. IEEE, 2010. DOI: <https://doi.org/10.1109/LICS.2010.48>.