



de Vrijer’s Strong Normalization in Haskell

Nachum Dershowitz

School of Computer Science and AI, Tel Aviv University, Ramat Aviv, Israel
emailnachum@tau.ac.il

Abstract

Roel de Vrijer’s assignment maps a simply typed λ -term to a measure whose numerical component decreases with each β -contraction. We present that assignment as a compact Haskell program in which object-language binders are implemented by Haskell binders, and the functional component of the measure is stored as an ordinary Haskell function, so that no separate environment-passing evaluator is needed in the main program. A proof that this is a decreasing bound on all subsequent reduction sequences is included. An accompanying Rocq development interprets intrinsically typed de Bruijn terms in the same measurement algebra and machine-checks substitution, contextual decrease, and strong normalization.

1 Introduction

A challenge, attributed to Kurt Gödel and William Howard, is to find a transparent ordinal-valued measure for simply typed terms that decreases every time a β -redex is contracted. The open-problem list for rewriting includes this question [11, 9, Problem 19], and it likewise appears as an “unsolved” problem in the typed λ -calculus (*TLCA*) list [13, Problem 26].

Strong normalization itself has many standard proofs, most notably via reducibility or related syntactic arguments [17, 22, 12, 15, 14, 19]. Roel de Vrijer’s solution [8], further explained in an unpublished note by Henk Barendregt [4], is significant because the size is computed together with a functional continuation of the same estimate: The measure of a function term carries along enough information to measure all future applications of that function term.

A Scheme implementation of de Vrijer’s measure was given at the termination workshop in 2021 [10]. Here it is recast in Haskell, a call-by-need language. Since the dot component can be stored directly as a Haskell function, the implementation eliminates the quoting and unquoting used in the Scheme version. Our implementation below goes further and lets Haskell binders carry the object-language environment through ordinary lexical scope.

De Vrijer’s finite-developments theorem is an earlier example of such a quantitative analysis of reduction: a natural-number annotation gives the length of a longest complete development [7]. Sørensen later treated the dual problem for shortest developments [20]. These results belong to the same broad tradition of measuring reductions directly. Other decreasing measures for simply typed β -reduction include Barenbaum and Sottile’s W - and T^m -measures [3] and Stirton’s proof-theoretic natural-number assignment for $\lambda\beta$ -reduction [21]. Schwichtenberg and Beckmann study bounds on the length of typed reduction sequences [18, 5]. There is also an unpublished Agda formalization of weak call-by-value reduction by Barenbaum [1]. Related uses of developments

and decreasing measures appear in Boudol’s intersection-type proof [6] and in later work by Barenbaum, Ronchi Della Rocca, and Sottile on idempotent intersection types [2].

The code archive¹ contains both this simple higher-order implementation and a named, environment-passing first-order implementation; only the simple implementation is described and proved correct here. It also contains a Rocq [23] (formerly Coq) development based on intrinsically typed de Bruijn syntax. Its evaluator folds that syntax into the same type-indexed algebra of measurements as the Haskell combinators. The development formalizes typed renaming and substitution, full contextual β -reduction, strict decrease of the measure, and strong normalization.²

2 The Measure

The construction has three layers. First, a measurement domain is assigned to every simple type. Second, two auxiliary operations propagate a numerical charge through those domains and provide canonical measurements at arbitrary types. Third, the three syntactic forms are interpreted in the resulting algebra.

Measurement domains. Terms and types are as usual. For every type τ we have a domain D_τ of semantic measurements, with $D_o = \mathbb{N}$ and $D_{\sigma \rightarrow \tau} = (D_\sigma \rightarrow D_\tau) \times \mathbb{N}$. A base-type measurement is a number; a function-type measurement $a = \langle f, n \rangle$ has *dot* part $a^\bullet = f$ and *star* part $a^* = n$. At base type, $n^* = n$. The star part is the current reduction budget; the dot part measures future applications. For example, $\langle f, 3 \rangle \in D_{o \rightarrow o}$ assigns budget 3 to the unapplied function and produces $f(n)$ when applied to an argument measured by n .

Lifting charges and canonical measurements. The semantic clauses use two operations. The first, add_k , lifts the addition of k through a measurement of any type:

$$\text{add}_k(n) = n + k, \quad \text{add}_k(\langle f, n \rangle) = \langle \lambda a. \text{add}_k(f(a)), n + k \rangle.$$

The charge is added both to the current star component and recursively to every future result of the dot component. This is essential at function type: a charge attached before an application must still be present after that application. At type $o \rightarrow o$, both the unapplied budget and every base-type result increase by k .

The second operation gives the canonical measurement of star value n at a specified type:

$$c_o(n) = n, \quad c_{\sigma \rightarrow \tau}(n) = \langle \lambda a. c_\tau(n + a^*), n \rangle.$$

At type $o \rightarrow o$ this specializes to $c_{o \rightarrow o}(n) = \langle \lambda a. n + a, n \rangle$. In particular, $c_{o \rightarrow o}(0) = \langle \lambda a. a, 0 \rangle$. Canonical measurements complete finitely written valuations and supply the probe for an abstraction’s star component. Valuations are written as finite maps: an unlisted x has value $\rho(x) = c_{\text{type}(x)}(0)$, making the valuation total. For a closed term evaluated from the empty map, every variable occurrence is supplied by its binder’s update. The probe $c_\sigma(0)$ measures an abstraction’s body before an argument is supplied.

¹At https://github.com/NachumDershowitz/SN_Haskell.

²ChatGPT 5.5 Pro assisted in developing the Haskell implementation, Rocq formalization, and proofs.

Interpretation of terms. Assuming that a valuation ρ maps each variable of type τ to an element of D_τ , define

$$\begin{aligned} \llbracket x \rrbracket_\rho &= \rho(x), \\ \llbracket MN \rrbracket_\rho &= \llbracket M \rrbracket_\rho^\bullet(\llbracket N \rrbracket_\rho), \\ \llbracket \lambda x^\sigma. M \rrbracket_\rho &= \langle \lambda a. \mathbf{add}_{a^*+1}(\llbracket M \rrbracket_{\rho[x \mapsto a]}), \llbracket M \rrbracket_{\rho[x \mapsto c_\sigma(0)]}^* \rangle. \end{aligned}$$

Here $\rho[x \mapsto a]$ denotes valuation update. Capture-avoiding substitution of N for x in M is written $M[x \mapsto N]$. For a closed term M , the computed measure is $\|M\| = \llbracket M \rrbracket_\emptyset^*$.

The abstraction's dot part evaluates the body with $x \mapsto a$ and adds $a^* + 1$: 1 pays for the outer β -contraction, while a^* accounts for reductions in the argument even when the body ignores it. Its star part records the body's budget with x assigned its canonical zero measurement.

The central decrease is already visible at a root redex. If $a = \llbracket N \rrbracket_\rho$, then

$$\llbracket (\lambda x^\sigma. M) N \rrbracket_\rho = \mathbf{add}_{a^*+1}(\llbracket M \rrbracket_{\rho[x \mapsto a]}). \quad (1)$$

The substitution lemma proved in Section 4 identifies the term inside **add** with $\llbracket M[x \mapsto N] \rrbracket_\rho$. Thus the redex has the measurement of its contractum plus a positive charge, lifted through the result type.

As a preliminary example, let $I = \lambda x^o. x$. At a base measurement n , its dot part is $\mathbf{add}_{n+1}(n) = 2n + 1$ and its star part is 0, so

$$\llbracket I \rrbracket_\emptyset = \langle \lambda n. 2n + 1, 0 \rangle. \quad (2)$$

If $y : o$ has measurement 0, then $Iy \rightarrow_\beta y$ and the corresponding measurements are $1 > 0$.

Example. Barendregt's note [4] contains ten hand-evaluated examples. For the term $E = (\lambda f^{o \rightarrow o}. \lambda x^o. f x) I$, our Scheme computation in [10] gives 2, whereas the value printed in that note is 1. By (2), $\llbracket I \rrbracket_\emptyset = \langle i, 0 \rangle$ with $i(n) = 2n + 1$. For the first contractum $J = \lambda x^o. Ix$, the dot part is $j(n) = \mathbf{add}_{n+1}(i(n)) = 3n + 2$ and the star part is $i(0) = 1$. Applying the outer abstraction of E to $\llbracket I \rrbracket_\emptyset$ adds $\llbracket I \rrbracket_\emptyset^* + 1 = 1$, giving

$$\llbracket J \rrbracket_\emptyset = \langle \lambda n. 3n + 2, 1 \rangle, \quad \llbracket E \rrbracket_\emptyset = \mathbf{add}_1(\llbracket J \rrbracket_\emptyset) = \langle \lambda n. 3n + 3, 2 \rangle.$$

Thus the star components follow the full reduction sequence exactly in this case:

$$\begin{array}{ccccc} (\lambda f. \lambda x. f x) I & \rightarrow_\beta & \lambda x. Ix & \rightarrow_\beta & I. \\ 2 & & 1 & & 0 \end{array}$$

3 The Simple Haskell Program

The Haskell file is compact and uses only the standard **base** library. Running it with the example outputs 2,³ the value calculated just above.

Types. An algebraic datatype is described by its constructors; a generalized algebraic datatype (GADT) specifies more precise constructor result types [16]. The empty Haskell type **Base** represents o , and the GADT **Ty** represents object types. The initial pragmas enable GADTs and the empty marker type.

³`runghc SN_Haskell1.hs.`

```

{-# LANGUAGE EmptyDataDecls #-}
{-# LANGUAGE GADTs #-}
{-# OPTIONS_GHC -Wall #-}

module Main where

import Numeric.Natural (Natural)

-- Object-language types, indexed by Haskell types.

data Base

data Ty a where
  TBase :: Ty Base
  TArr  :: Ty a -> Ty b -> Ty (a -> b)

```

Measures. The datatype `Meas` mirrors the domains D_τ . Constructor `N` stores a base-type natural number. Constructor `F` stores a Haskell function for the dot component and a natural number for the star component. The definitions of `star`, `dot`, `add`, and `canon` are direct translations of Section 2.

```

-- Type-indexed de Vrijer measurements.

data Meas a where
  N :: Natural -> Meas Base
  F :: (Meas a -> Meas b) -> Natural -> Meas (a -> b)

star :: Meas a -> Natural
star (N n)    = n
star (F _ n)  = n

dot :: Meas (a -> b) -> Meas a -> Meas b
dot (F f _) a = f a

add :: Natural -> Meas a -> Meas a
add k (N n)    = N (n + k)
add k (F f n)  = F (\a -> add k (f a)) (n + k)

canon :: Ty a -> Natural -> Meas a
canon TBase n    = N n
canon (TArr _ b) n = F (\a -> canon b (n + star a)) n

```

GHC, the Glasgow Haskell Compiler, uses the indices of `Ty` (above) and `Meas` and the type of `app` (below) to rule out ill-typed applications.

Terms. The line `type Term a = Meas a` is the key simplification. An object term is represented by its measurement directly. Variables are host-language variables, application is `dot`, and abstraction is de Vrijer's operation. The Haskell closure created by `lam` carries the environment that our first-order implementation passes explicitly.

```

-- Higher-order/final representation of terms.
-- Object binders are Haskell binders, so Haskell closures carry the
-- environment that the first-order version passes explicitly.

type Term a = Meas a

var :: Meas a -> Term a
var = id

app :: Term (a -> b) -> Term a -> Term b
app = dot

lam :: Ty a -> (Term a -> Term b) -> Term (a -> b)
lam ty body =
  F (\a -> add (star a + 1) (body a))
    (star (body (canon ty 0)))

measure :: Term a -> Natural
measure = star

```

Example. The previous example is represented by ordinary Haskell application and binders:

```

-- Example: (\f{o -> o}. \x{o. f x) (\x{o. x)

base :: Ty Base
base = TBase

baseToBase :: Ty (Base -> Base)
baseToBase = TArr base base

ex :: Term (Base -> Base)
ex = app (lam baseToBase (\f -> lam base (\x -> app f x)))
        (lam base (\x -> x))

main :: IO ()
main = print (measure ex)

```

The two object-level binders printed as x in the mathematical notation are distinct Haskell binders with separate lexical scopes. The value `measure ex` is 2. The checker in the archive verifies the reported star values; when an example specifies a functional component, it compares that function on the inputs 0..4.

The code also contains a first-order named version and a trace program.

4 Proof of Correctness

The proof follows the structure of the above construction. We first isolate the ordered, monotone part of each measurement domain; then show that `add`, `c`, and term interpretation preserve it; establish substitution; and finally lift root decrease through every reduction context.

Definition 1 (Orders and admissibility). For each type τ , define a non-strict quasi-order $\geq_\tau$, a strict order $>_\tau$, and a predicate of *admissibility* on D_τ . At base type, the orders are the usual orders on $\mathbb{N}$, and every natural number is admissible. At function type, with both function comparisons quantified over all admissible $a \in D_\sigma$,

$$\begin{aligned}
\langle f, m \rangle &\geq_{\sigma \rightarrow \tau} \langle g, n \rangle && \text{iff } m \geq n \text{ and } f(a) \geq_\tau g(a), \\
\langle f, m \rangle &>_{\sigma \rightarrow \tau} \langle g, n \rangle && \text{iff } m > n \text{ and } f(a) >_\tau g(a).
\end{aligned}$$

A function-type measure $\langle f, n \rangle \in D_{\sigma \rightarrow \tau}$ is admissible when $f(a)$ is admissible for every admissible $a \in D_\sigma$, and, for all admissible $a, b \in D_\sigma$, f is monotone for both comparisons: $a \geq_\sigma b \Rightarrow f(a) \geq_\tau f(b)$ and $a >_\sigma b \Rightarrow f(a) >_\tau f(b)$. A valuation is admissible when it maps each variable to an admissible measure of the variable's type. For valuations ρ and ρ' , write $\rho \geq \rho'$ when $\rho(x) \geq_{\text{type}(x)} \rho'(x)$ for every variable x .

The two pointwise properties have distinct roles in contextual reduction. When the function position decreases from $\langle f, m \rangle$ to $\langle g, n \rangle$, the definition of $>_{\sigma \rightarrow \tau}$ gives $f(a) > g(a)$ at every admissible argument a . When the argument position decreases from a to b , admissibility of the unchanged function measurement gives $f(a) > f(b)$. Admissibility therefore identifies the part of the full function space that preserves reductions of its arguments.

An admissible erasing function. Let $K_y = \lambda x^o. y$, where $y, z : o$ and $\rho(y) = \rho(z) = 0$. Although the body ignores its argument, the abstraction clause gives $\llbracket K_y \rrbracket_\rho = \langle \lambda n. n + 1, 0 \rangle$, whose dot part is strictly increasing. With $I = \lambda x^o. x$, the sequence $K_y(Iz) \rightarrow_\beta K_y z \rightarrow_\beta y$ has star values $2 > 1 > 0$. Thus the charge for the argument records reductions that can take place before that argument is discarded. If the abstraction clause added only 1, omitting the argument's star component, the first two measurements would both be 1 and strict decrease in the argument position would fail.

A counterexample shows what would happen were every measurement declared admissible. Let $x : o \rightarrow o$ and $y : o$, put $\rho(y) = 0$, and assign $\rho(x) = \langle h, 0 \rangle$, where $h(0) = 1$ and $h(n+1) = 0$. For $Q = (\lambda z^o. z)y$, we have $\llbracket Q \rrbracket_\rho = 1 > 0 = \llbracket y \rrbracket_\rho$. Yet $xQ \rightarrow_\beta xy$ while $\llbracket xQ \rrbracket_\rho = h(1) = 0 < 1 = h(0) = \llbracket xy \rrbracket_\rho$. Thus h reverses the strict comparison and $\langle h, 0 \rangle$ is not admissible. Canonical measurements and denotations under admissible valuations satisfy the conditions, as proved below.

The strict order is contained in the non-strict order, and $a >_\tau b$ always implies $a^* > b^*$. It is therefore well founded on admissible measurements, since every strict descent lowers the natural-number star component.

Next we relate our simple Haskell representation to de Vrijer's equations. A Haskell valuation η assigns a value of type **Meas** $\mathbf{a}$ to each source variable of the corresponding object type. As for mathematical valuations, an unlisted variable $x : \sigma$ receives its canonical zero measurement, here $\text{canon } \hat{\sigma} 0$. The value represented by a term is defined recursively by

$$\begin{aligned} \mathcal{H}_\eta(x) &= \text{var } (\eta(x)), \\ \mathcal{H}_\eta(MN) &= \text{app } (\mathcal{H}_\eta(M)) (\mathcal{H}_\eta(N)), \\ \mathcal{H}_\eta(\lambda x^\sigma. M) &= \text{lam } \hat{\sigma} (\lambda p. \mathcal{H}_{\eta[x \mapsto p]}(M)), \end{aligned}$$

where $\hat{\sigma}$ is the Haskell witness of the object type σ . Let $\iota : \mathbb{N} \rightarrow \mathbf{Natural}$ be the canonical representation of natural numbers in Haskell. Compare Haskell values with mathematical measurements by the type-indexed relation

$$\begin{aligned} Nz \simeq_o n &\iff z = \iota n, \\ Ffz \simeq_{\sigma \rightarrow \tau} \langle g, n \rangle &\iff z = \iota n \text{ and } (p \simeq_\sigma a \Rightarrow f(p) \simeq_\tau g(a)) \text{ for all } p, a. \end{aligned}$$

Write $\eta \simeq \rho$ when corresponding entries of the two valuations are related in this sense.

Proposition 2 (Program implements the equations). *For every well-typed term $M : \tau$, every typed mathematical valuation ρ , and every Haskell valuation $\eta \simeq \rho$, we have $\mathcal{H}_\eta(M) \simeq_\tau \llbracket M \rrbracket_\rho$. Consequently, for every closed term M ,*

$$\text{measure}(\mathcal{H}_\emptyset(M)) = \iota(\llbracket M \rrbracket_\emptyset^*). \quad (3)$$

Proof. Induction on types shows that the two implementations of the auxiliary operations respect $\simeq$: if $p \simeq_\tau a$, then $\mathbf{add}(\iota k) p \simeq_\tau \mathbf{add}_k(a)$; also $\mathbf{canon} \hat{\tau}(\iota n) \simeq_\tau c_\tau(n)$. Now induct structurally on M . The variable case is $\mathbf{var} = \mathbf{id}$, and the application case is $\mathbf{app} = \mathbf{dot}$. For an abstraction, fix arbitrary related arguments $p \simeq_\sigma a$. The extended valuations $\eta[x \mapsto p]$ and $\rho[x \mapsto a]$ correspond, so the induction hypothesis for the body, followed by compatibility of $\mathbf{add}$, gives the related dot components. Applying the same induction hypothesis at the related canonical measurements gives the related star components. This proves the function-type clause of $\simeq$. Equation (3) follows from $\mathbf{measure} = \mathbf{star}$. $\square$

Some elementary facts are used repeatedly in the termination proof.

Lemma 3 (Auxiliary operations). *Let τ be a type. For all admissible $a, b \in D_\tau$ and all $k, \ell \in \mathbb{N}$,*

$$\mathbf{add}_k(a) \text{ is admissible}, \quad (4)$$

$$\mathbf{add}_k(a)^* = a^* + k, \quad (5)$$

$$a \geq_\tau b \text{ and } k \geq \ell \implies \mathbf{add}_k(a) \geq_\tau \mathbf{add}_\ell(b), \quad (6)$$

$$a \geq_\tau b, \ k \geq \ell, \text{ and } (a >_\tau b \text{ or } k > \ell) \implies \mathbf{add}_k(a) >_\tau \mathbf{add}_\ell(b), \quad (7)$$

$$k > 0 \implies \mathbf{add}_k(a) >_\tau a. \quad (8)$$

Moreover, for all $m, n \in \mathbb{N}$,

$$c_\tau(n) \text{ is admissible}, \quad (9)$$

$$c_\tau(n)^* = n, \quad (10)$$

$$n \geq m \implies c_\tau(n) \geq_\tau c_\tau(m), \quad (11)$$

$$n > m \implies c_\tau(n) >_\tau c_\tau(m). \quad (12)$$

Proof. Induct on τ . At base type, Equations (4)–(12) reduce to elementary arithmetic. Suppose $\tau = \sigma \rightarrow v$, and write $a = \langle f, r \rangle$ and $b = \langle g, s \rangle$. Equation (5) and the star part of Equations (6) and (7) follow immediately from the definitions and the corresponding arithmetic comparisons $r + k \geq s + \ell$ or $r + k > s + \ell$.

For Eq. (4), admissibility of a gives admissibility of $f(c)$ for every admissible c ; the induction hypothesis for (4) then gives admissibility of $\mathbf{add}_k(f(c))$. For admissible c, d with $c \geq d$, admissibility of a gives $f(c) \geq f(d)$, and the induction hypothesis for (6), with the same k on both sides, gives $\mathbf{add}_k(f(c)) \geq \mathbf{add}_k(f(d))$. The strict-monotonicity clause follows in the same way from the induction hypothesis for (7). Thus $\mathbf{add}_k(a)$ is admissible.

For Eq. (6), the function-type hypothesis $a \geq b$ gives $f(c) \geq g(c)$ at every admissible c ; applying the induction hypothesis for (6) pointwise yields $\mathbf{add}_k(f(c)) \geq \mathbf{add}_\ell(g(c))$. Equation (7) is obtained pointwise from its induction hypothesis: strictness comes from $f(c) > g(c)$ when $a > b$, or from $k > \ell$ otherwise. Equation (8) follows from (7) with $b = a$ and $\ell = 0$, using the reflexivity of $\geq_\tau$ on admissible measurements and the identity $\mathbf{add}_0(a) = a$.

For canonical measurements, Eq. (10) is immediate. At an admissible argument c , the dot component of $c_{\sigma \rightarrow v}(n)$ is $c_v(n + c^*)$, which is admissible by the induction hypothesis for (9). If $c \geq d$, then $c^* \geq d^*$, so the induction hypothesis for (11) gives non-strict monotonicity of the dot component; if $c > d$, the corresponding star components are strictly ordered, and (12) gives strict monotonicity. This proves (9). Finally, Equations (11) and (12) follow pointwise from their induction hypotheses at v , since addition of the common value c^* preserves the comparison between n and m . $\square$

Lemma 4 (Representation invariant). *If $M : \tau$ and ρ is admissible, then $\llbracket M \rrbracket_\rho$ is an admissible element of D_τ . In addition, $\llbracket M \rrbracket_\rho$ is monotone in ρ : if admissible valuations $\rho \geq \rho'$ pointwise, then $\llbracket M \rrbracket_\rho \geq_\tau \llbracket M \rrbracket_{\rho'}$.*

Proof. By induction on the typing derivation. For variables, lookup returns either the assigned admissible value or a canonical one, which is admissible by (9). For an application MN , the induction hypothesis gives an admissible function measurement for M and an admissible argument measurement for N ; admissibility of the function measurement gives admissibility of $\llbracket MN \rrbracket_\rho$. For monotonicity of an application, write $\llbracket M \rrbracket_\rho = \langle f, m \rangle$, $\llbracket M \rrbracket_{\rho'} = \langle g, n \rangle$, $a = \llbracket N \rrbracket_\rho$, and $b = \llbracket N \rrbracket_{\rho'}$. The induction hypothesis gives $\langle f, m \rangle \geq \langle g, n \rangle$ and $a \geq b$. Hence $f(a) \geq g(a)$ by the function-type order and $g(a) \geq g(b)$ by admissibility of $\langle g, n \rangle$.

For an abstraction $\lambda x^\sigma.M$, the star component is a natural number by induction under $\rho[x \mapsto c_\sigma(0)]$. For an admissible argument a , induction under $\rho[x \mapsto a]$ and Eq. (4) show that $\text{add}_{a^*+1}(\llbracket M \rrbracket_{\rho[x \mapsto a]})$ is admissible. For admissible $a \geq b$, we have $\rho[x \mapsto a] \geq \rho[x \mapsto b]$ and $a^* + 1 \geq b^* + 1$; Eq. (6) gives non-strict monotonicity of the dot part. If $a > b$, the body comparison is non-strict and the numerical parameter is strict, so Eq. (7) gives strict monotonicity. Monotonicity in ρ is obtained by applying the induction hypothesis to the body under the corresponding extended valuations, both for the dot component and for the star component. $\square$

Lemma 5 (Substitution). *Let M and N be well typed, with $\text{type}(N) = \text{type}(x)$, and suppose substitution is capture-avoiding. Then, extensionally,*

$$\llbracket M[x \mapsto N] \rrbracket_\rho = \llbracket M \rrbracket_{\rho[x \mapsto \llbracket N \rrbracket_\rho]}.$$

Proof. Induct on the structure of M , proving simultaneously the usual weakening property: changing a valuation at a variable not free in a term does not change that term's denotation. The variable cases are the two lookup cases. The application case follows by applying the induction hypotheses to both subterms. For an abstraction, α -convert first so that the binder y is different from x and does not occur free in N . For the dot component, fix an arbitrary argument a , extend both valuations with $y \mapsto a$, and apply the induction hypothesis to the body. For the star component, extend both valuations with $y \mapsto c_{\text{type}(y)}(0)$ and apply the induction hypothesis again. Since $y \notin \text{FV}(N)$, weakening keeps $\llbracket N \rrbracket_\rho$ unchanged under the added y -binding, and the two updates commute. $\square$

Lemma 6 (Reduction decreases the measure). *If $M \rightarrow_\beta N$ and both terms have type τ , then every admissible valuation ρ satisfies $\llbracket M \rrbracket_\rho >_\tau \llbracket N \rrbracket_\rho$.*

Proof. Induct on the reduction context. At the root, $(\lambda x^\sigma.P)Q \rightarrow_\beta P[x \mapsto Q]$. Let $a = \llbracket Q \rrbracket_\rho$. Equation (1) gives $\llbracket (\lambda x.P)Q \rrbracket_\rho = \text{add}_{a^*+1}(\llbracket P \rrbracket_{\rho[x \mapsto a]})$. By Lemma 4, a is admissible, so $\rho[x \mapsto a]$ is an admissible valuation; another use of Lemma 4 gives admissibility of $\llbracket P \rrbracket_{\rho[x \mapsto a]}$. By Lemma 5, $\llbracket P \rrbracket_{\rho[x \mapsto a]} = \llbracket P[x \mapsto Q] \rrbracket_\rho$. Since $a^* + 1 > 0$, (8) gives the strict inequality for the contracted redex.

For a reduction in the function position, $PQ \rightarrow_\beta P'Q$ with $P \rightarrow_\beta P'$. The induction hypothesis gives $\llbracket P \rrbracket_\rho > \llbracket P' \rrbracket_\rho$. Writing these function measurements as $\langle f, m \rangle$ and $\langle g, n \rangle$, and setting $a = \llbracket Q \rrbracket_\rho$, Lemma 4 gives admissibility of a . The definition of $>_{\sigma \rightarrow \tau}$ then gives $f(a) > g(a)$, which is the required comparison of applications. For a reduction in the argument, $PQ \rightarrow_\beta PQ'$, induction gives $\llbracket Q \rrbracket_\rho > \llbracket Q' \rrbracket_\rho$; Lemma 4 gives admissibility of $\llbracket P \rrbracket_\rho$, so its dot part is strictly monotone.

For a reduction below a lambda, $\lambda x.P \rightarrow_\beta \lambda x.P'$, compare star components by applying the induction hypothesis to the body under the admissible valuation $\rho[x \mapsto c_\sigma(0)]$. For each

admissible argument a , compare dot components by applying the induction hypothesis under the admissible valuation $\rho[x \mapsto a]$, and then use (7) with equal numerical parameters to preserve strictness under add_{a^*+1} . Thus the abstraction measurements are strictly ordered. $\square$

Theorem 7 (Correctness of `measure`). *For every closed, well-typed source term M , the value $\text{measure}(\mathcal{H}_\emptyset(M))$ is a natural number. If $M \rightarrow_\beta N$, then*

$$\text{measure}(\mathcal{H}_\emptyset(M)) > \text{measure}(\mathcal{H}_\emptyset(N)). \quad (13)$$

Therefore, the length of a β -reduction sequence starting from M is at most the initial value.

Proof. Equation (3) identifies the program's output with the `Natural` representation of $\llbracket M \rrbracket_\emptyset^*$. The empty valuation is admissible. If $M \rightarrow_\beta N$, subject reduction gives that N has the same type as M , and N remains closed. Lemma 6 gives $\llbracket M \rrbracket_\emptyset > \llbracket N \rrbracket_\emptyset$ and therefore $\llbracket M \rrbracket_\emptyset^* > \llbracket N \rrbracket_\emptyset^*$. With Eq. (3), this proves (13). The argument repeats at each subsequent step; every step lowers the natural number by at least one, so the initial value bounds the length of the sequence. $\square$

Interpreting the output. For a closed, well-typed term M , $\|M\| = 0$ exactly when M is β -normal. Under the canonical-zero valuation, a neutral normal form $xN_1 \cdots N_k : \tau$ has measurement $c_\tau(0)$: each normal argument has star component 0. Simultaneous induction on neutral and normal forms proves this claim and shows that an abstraction has star component 0 by applying the induction hypothesis to its body with the bound variable assigned its canonical zero. Conversely, Lemma 6 gives a strictly positive star component to every term that admits a reduction step. The positive output is a bound rather than necessarily the exact length. For example, with $I = \lambda x^o.x$, the sequence $\lambda y^o.I(Iy) \rightarrow_\beta \lambda y^o.Iy \rightarrow_\beta \lambda y^o.y$ has measures $3 > 1 > 0$, although every complete reduction sequence from the first term has length 2. De Vrijer also gives an exact valuation whose star component equals the length of a longest β -reduction sequence [8, Sections 2.3 and 4]. The present implementation uses his simpler loose valuation [8, Section 2.4].

The Rocq formalization. The file `SN_Haskell_Rocq.v` formalizes the measurement algebra, typed syntax, substitution, contextual reduction, and the normalization argument above.⁴ Every lemma and theorem in the development has a complete proof. Equality of function components is handled extensionally using Rocq's standard functional-extensionality principle.

The common measurement algebra is represented by Haskell's `Meas a` and Rocq's `den t`. The operations `star`, `dot`, `add`, and `canon` have the same names in both programs; Rocq's `appD` and `lamD` implement Haskell's `app` and `lam`. The type synonym `Term a = Meas a` makes the Haskell representation final: constructing a term with the combinators computes its measurement. Rocq instead folds explicit syntax into this algebra. Haskell's `Natural` and Rocq's `nat` correspond to $\mathbb{N}$ through ι and $\simeq_o$.

Intrinsically typed de Bruijn syntax. The Rocq families `var Gamma t` and `tm Gamma t` represent a de Bruijn variable and a term of type t in context Γ . The constructors `TVar`, `TApp`, and `TLam` enforce typing in their indices. A valuation in `Val Gamma` assigns each typed index a measurement of the corresponding type. The evaluator is the fold from this syntax into the common measurement algebra:

$$\begin{aligned} \text{eval}_\rho(\text{TVar } i) &= \rho(i), \\ \text{eval}_\rho(\text{TApp } M \ N) &= \text{appD}(\text{eval}_\rho(M), \text{eval}_\rho(N)), \\ \text{eval}_\rho(\text{TLam } M) &= \text{lamD}(\lambda a. \text{eval}_{a::\rho}(M)). \end{aligned}$$

⁴With Rocq 9.2.0 [23], run `rocq compile SN_Haskell_Rocq.v`.

Here $a :: \rho$ is the Rocq operation `val_ext`, which assigns a to de Bruijn index 0 and shifts the previous valuation to the remaining indices.

Fix an ordered context $\Gamma = (x_0 : \tau_0, \dots, x_{r-1} : \tau_{r-1})$ of distinct variables, with x_0 represented by de Bruijn index 0. After the usual α -renaming of binders, db_Γ maps x_i to its typed index, maps application componentwise, and maps $\lambda x^\sigma.M$ to $\text{TLam}(\text{db}_{(x:\sigma,\Gamma)}(M))$. For a named valuation ρ , the indexed valuation $\bar{\rho}$ maps the index of x_i to $\rho(x_i)$; under a binder it is extended by `val_ext`.

For the running example, $\text{db}_\varnothing(E) = (\lambda^{\circ \rightarrow \circ}.\lambda^\circ.\underline{1}\underline{0})(\lambda^\circ.\underline{0})$. In the doubly nested body, indices $\underline{0}$ and $\underline{1}$ denote x and f , represented by `TVar vz` and `TVar (vs vz)`. The resulting Rocq term is `ex_tm`; `ex_star` proves that its star component is 2, agreeing with `measure ex`.

Proposition 8 (Haskell/de Bruijn correspondence). *Let $M : \tau$ be well typed in Γ , let ρ be any typed mathematical valuation, let $\eta \simeq \rho$ be a corresponding Haskell valuation, and let $\bar{\rho}$ be the indexed Rocq valuation just defined. Then $\mathcal{H}_\eta(M) \simeq_\tau \llbracket M \rrbracket_\rho = \text{eval}_{\bar{\rho}}(\text{db}_\Gamma(M))$, where the equality uses the recursive identification of D_τ with the Rocq carrier `den t`. For a closed term, taking star components gives $\text{measure}(\mathcal{H}_\varnothing(M)) = \iota(\text{star}(\text{eval}_\varnothing(\text{db}_\varnothing(M))))$.*

Proof. By Proposition 2, $\mathcal{H}_\eta(M) \simeq_\tau \llbracket M \rrbracket_\rho$. It remains to prove $\llbracket M \rrbracket_\rho = \text{eval}_{\bar{\rho}}(\text{db}_\Gamma(M))$, which we do by structural induction on M . A named variable and its de Bruijn index select corresponding entries of ρ and $\bar{\rho}$. Application uses the same dot operation on both sides. For an abstraction, extend the two valuations by an arbitrary argument measurement and apply the induction hypothesis to the body; apply it again at $c_\sigma(0)$ for the star component. The clauses for `lamD` and the mathematical abstraction operation then agree, extensionally at arrow type. The closed-term equation follows by taking star components and using Eq. (3). $\square$

Proposition 8 is the structural bridge. Starting from de Bruijn syntax, Rocq proves the representation invariant in `eval_adm_ge` and substitution in `eval_subst0`, whose equation is $\text{eval}_\rho(\text{subst0 } N \ M) = \text{eval}_{\text{eval}_\rho(N)::\rho}(M)$. The four constructors of `step` cover root contraction, either application position, and reduction under a lambda. The theorem `step_decreases` proves Lemma 6; `star_decreases` projects it to natural numbers. Well-founded induction on this component gives accessibility in `strong_normalization`; `strong_normalization_all` uses canonical valuations and `strong_normalization_closed` specializes to closed terms. The length bound follows by iterating one-step descent. A declaration-by-declaration table accompanies the sources as `CORRESPONDENCE.md`.

5 Conclusion

Our Haskell implementation follows de Vrijer's construction, using host-language functions for dot components and binders for the object-language environment. This eliminates the explicit environment parameter and the quoting and unquoting of our earlier Scheme implementation [10]. The proof establishes that the computed star component is in fact a decreasing natural-number bound for full simply typed β -reduction. The complementary Rocq evaluator folds intrinsically typed de Bruijn syntax into the same algebra and formalizes substitution, contextual decrease, and strong normalization.

The archive for this work includes the simple higher-order Haskell program, named and tracing versions, an example checker, and the Rocq development.

References

- [1] Pablo Barenbaum. Strong normalization of the simply typed lambda-calculus in Agda. GitHub Gist. Unpublished formalization based on de Vrijer's increasing-functional proof. URL: <https://gist.github.com/foones/50ac4eb365e35f4571516558055ccd6e>.
- [2] Pablo Barenbaum, Simona Ronchi Della Rocca, and Cristian Sottile. Strong normalization through idempotent intersection types: A new syntactical approach. *Electronic Notes in Theoretical Informatics and Computer Science*, 5, 2025. Proceedings of MFPS XLI; entics:16693. URL: <https://arxiv.org/abs/2503.09831>, doi:10.46298/entics.16693.
- [3] Pablo Barenbaum and Cristian Sottile. Two decreasing measures for simply typed λ -terms. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023)*, volume 260 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:19, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.FSCD.2023.11>, doi:10.4230/LIPIcs.FSCD.2023.11.
- [4] Henk Barendregt. Digesting the proof of Roel de Vrijer that $\lambda_{\rightarrow} \models \text{SN}$. Unpublished note, April 2019. URL: <https://www.cs.tau.ac.il/~nachumd/Henk.pdf>.
- [5] Arnold Beckmann. Exact bounds for lengths of reductions in typed λ -calculus. *The Journal of Symbolic Logic*, 66(3):1277–1285, 2001. doi:10.2307/2695106.
- [6] Gérard Boudol. On strong normalization in the intersection type discipline. In Martin Hofmann, editor, *Typed Lambda Calculi and Applications*, volume 2701 of *Lecture Notes in Computer Science*, pages 60–74, Berlin, Heidelberg, 2003. Springer. doi:10.1007/3-540-44904-3_5.
- [7] Roel C. de Vrijer. A direct proof of the finite developments theorem. *The Journal of Symbolic Logic*, 50(2):339–343, 1985. doi:10.2307/2274219.
- [8] Roel C. de Vrijer. Exactly estimating functionals and strong normalization. *Indagationes Mathematicae (Proceedings)*, 90(4):479–493, 1987. doi:10.1016/1385-7258(87)90012-6.
- [9] Nachum Dershowitz. The RTA list of open problems, 2009–2021. URL: <https://www.cs.tau.ac.il/~nachum/rtaloop/>.
- [10] Nachum Dershowitz. de Vrijer's measure for SN of $\lambda^{\rightarrow}$ in Scheme. In Samir Genaim, editor, *Proceedings of the 17th International Workshop on Termination (WST 2021)*, pages 9–14, July 2021. Affiliated with CADE-28, virtual event. URL: https://costa.fdi.ucm.es/wst2021/WST2021_proceedings.pdf.
- [11] Nachum Dershowitz, Jean-Pierre Jouannaud, and Jan Willem Klop. Open problems in rewriting. In R. Book, editor, *Proceedings of the Fourth International Conference on Rewriting Techniques and Applications*, volume 488 of *Lecture Notes in Computer Science*, pages 445–456, Berlin, Heidelberg, April 1991. Springer. doi:10.1007/3-540-53904-2_120.
- [12] Robin O. Gandy. Proofs of strong normalization. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 457–477. Academic Press, 1980.
- [13] Ryu Hasegawa, Luca Paolini, and Paweł Urzyczyn. TLCA list of open problems, July 2014. URL: <https://tlca.di.unito.it/oplclca/>.
- [14] Assaf J. Kfoury and Joe B. Wells. New notions of reduction and non-semantic proofs of strong β -normalization in typed λ -calculi. In *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, pages 311–321. IEEE Computer Society, 1995. doi:10.1109/LICS.1995.523266.
- [15] Jan Willem Klop. *Combinatory Reduction Systems*. PhD thesis, Utrecht University, Utrecht, 1980. Mathematical Centre Tracts 127, Mathematisch Centrum, Amsterdam.
- [16] Simon Peyton Jones, Dimitrios Vytiniotis, Stephanie Weirich, and Geoffrey Washburn. Simple unification-based type inference for GADTs. In *Proceedings of the 11th ACM SIGPLAN International Conference on Functional Programming*, pages 50–61. ACM, 2006. doi:10.1145/1159803.1159811.

- [17] Luis Elpidio Sanchis. Functionals defined by recursion. *Notre Dame Journal of Formal Logic*, 8(3):161–174, July 1967. doi:[10.1305/ndjfl/1093956080](https://doi.org/10.1305/ndjfl/1093956080).
- [18] Helmut Schwichtenberg. An upper bound for reduction sequences in the typed λ -calculus. *Archive for Mathematical Logic*, 30(5–6):405–408, 1991. doi:[10.1007/BF01621476](https://doi.org/10.1007/BF01621476).
- [19] Morten Heine Sørensen. Strong normalization from weak normalization in typed λ -calculi. *Information and Computation*, 133(1):35–71, 1997. doi:[10.1006/inco.1996.2622](https://doi.org/10.1006/inco.1996.2622).
- [20] Morten Heine Sørensen. A note on shortest developments. *Logical Methods in Computer Science*, 3(4:2):1–8, 2007. URL: <https://lmcs.episciences.org/838>, arXiv:0708.0200, doi:[10.2168/LMCS-3\(4:2\)2007](https://doi.org/10.2168/LMCS-3(4:2)2007).
- [21] William R. Stirton. Proof-theoretic investigation of $\lambda\beta$ -reduction in the simply typed λ -calculus. *Annals of Pure and Applied Logic*, 177(1):103657, 2026. doi:[10.1016/j.apal.2025.103657](https://doi.org/10.1016/j.apal.2025.103657).
- [22] William W. Tait. A realizability interpretation of the theory of species. In Rohit Parikh, editor, *Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 240–251. Springer, Berlin, Heidelberg, 1975. doi:[10.1007/BFb0064875](https://doi.org/10.1007/BFb0064875).
- [23] The Rocq Development Team. The Rocq prover, version 9.2.0. Software release and reference manual, 2026. URL: <https://rocq-prover.org/releases/9.2.0>.