Hereditary Substitution by Canonical Evaluation (SbE)

[Technical Report]

Larry Diehl and Tim Sheard

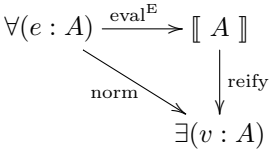
Portland State University
{ldiehl,sheard}@cs.pdx.edu

Abstract. Hereditary substitution is a version of substitution that works with canonical terms. Substituting into a canonical term may temporarily create a β-redex. Hereditary substitution immediately evaluates the redex to once again obtain a canonical term. Termination of hereditary substitution has primarily been studied *proof-theoretically*. However, formal proof-theoretic termination arguments of hereditary substitution have trouble scaling to systems with inductive types, such as Gödel's System T.
We present a *model-theoretic* termination argument of hereditary substitution that scales to systems with inductive types. Specifically, we adapt some of the techniques used in formalizing Normalization by Evaluation (NbE). Much of the existing NbE machinery can be reused. In some sense, we merely point out a beautiful coincidence: Canonical evaluation (evaluation from NbE, but defined on canonical terms) is a terminating model-theoretic definition of hereditary substitution. All of our work has been verified by Agda. [1]

Keywords: Hereditary substitution; normalization by evaluation; termination; type theory; formalization.

1 Introduction

Normalization by Evaluation (NbE) [4,6,7] is a model-theoretic technique for defining the semantics of a λ-calculus. For example, normalization for a total λ-calculus with inductive types (such as Gödel's System T [11]) can be defined by first evaluating a syntactic expression (possibly containing β-redexes) to a semantic value of a Kripke model [15,19], and then reifying the result to a syntactic value (not containing any β-redexes).

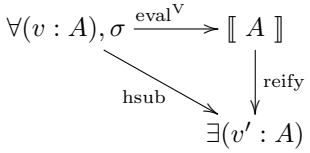


[1] The accompanying source code can be found at
https://github.com/larrytheliquid/sbe

NbE is especially convenient for formalizing the semantics of total λ-calculi in dependently typed languages (such as Coq [25], Agda [22], or Idris [5]), because the proof of termination of the semantics is *implicit* in its definition. In other words, normalization can be defined as a total function within a language based on Martin-Löf Type Theory [17,18,21]. In contrast, formalizing the semantics of expressions as a relation (e.g. a small or big-step relation), or as a coinductive function [3], requires the additional overhead of supplying an *explicit* termination proof *a posteriori* using techniques such as logical relations [23].

In a standard type theory *expressions* are emphasized and their semantics is defined directly. Alternatively, a *canonical type theory* emphasizes *values* (canonical terms) and defines the semantics of expressions in terms of hereditary substitution of values (for example, see Appendix D). Values are not closed under ordinary substitution, because a substitution can introduce a β-redex. Hereditary substitution fixes this problem by reducing whenever a β-redex is created as the result of substitution. But, it is not easy to show that traditional proof-theoretic definitions of hereditary substitution terminate (for a suitably complex λ-calculus, see Section 1.1 below) in dependently typed languages.

Our contribution is adapting NbE to canonical terms, giving us a model-theoretic definition of hereditary substitution. We call this technique "Hereditary Substitution by Canonical Evaluation (SbE)". Defining SbE requires us to introduce novel definitions, but we also get to reuse the model from NbE, as well as some of its definitions. SbE is defined by *canonically evaluating* a syntactic value to a semantic value (of a Kripke model), and then reifying back to a syntactic value.



SbE is a functional definition of hereditary substitution that contains an intrinsic termination proof, just like the functional definition of normalization that you get from NbE. Thus, SbE is amenable to formalization, allowing canonical type theory for λ-calculi with inductive types to be defined in dependent type theory. For simplicity, we use Gödel's System T for our examples in this paper, but SbE scales to systems with parameterized inductive types (such as lists or trees) for the same reasons that NbE does.

1.1 Proof Theoretic Semantics

Hereditary substitution has historically been studied in the context of proof-theoretic semantics. A proof-theoretic semantics takes syntactic inputs directly to syntactic outputs, without going through an intermediate semantic model. Additionally, a proof-theoretic termination proof (which can be implicit in the semantics or given explicitly) is completely syntactic (i.e. it does not use a logical

relation). Formalizing this approach in dependently typed languages has been limited to simple calculi. For example, Keller and Altenkirch [14] show how to formalize the semantics of the STLC in Agda using syntactic hereditary substitution. The termination of the hereditary substitution function is witnessed by lexicographic induction on terms and types, but this approach does not scale to λ-calculi with inductive types.

An advantage of the syntactic approach to hereditary substitution is that it demands less from the metalanguage than a semantic approach (i.e. the metalanguage function space is not needed to construct a model). A disadvantage of the syntactic approach is that exposing the lexicographic termination argument requires changing the structure of values, i.e. requiring values to be in spine form.

1.2 Outline

After a brief discussion about our notation, the rest of the paper is structured as follows:

- **Section 2** We review the model and theorems used by NbE, and explain the definition of NbE (and its implicit termination proof). Then, we present the novel definition of SbE, in which we reuse the model and some theorems from NbE. Finally, we compare and contrast SbE with NbE.
- **Section 3** We present the novel definition of canonical evaluation. In NbE, evaluation takes a syntactic expression and a semantic environment to a semantic value. Canonical evaluation is like evaluation, except it operates on values instead of expressions. In SbE, canonical evaluation takes a syntactic value and a semantic environment to a semantic value.
- **Section 4** We present the novel definition of environment reflection. Because hereditary substitution already has a syntactic environment parameter (the "substitution" to be applied to the variables of the syntactic value that we are hereditarily substituting into), we must reflect that syntactic environment as a semantic environment so that we may use it as an argument of canonical evaluation.
- **Section 5** We discuss related and future work. In particular, we compare and contrast proof-theoretic and model-theoretic hereditary substitution in more detail. We also point out how SbE (model-theoretic hereditary substitution) can be adapted from NbE to calculi with inductive types. In contrast, adapting proof-theoretic hereditary substitution to calculi with inductive types has been problematic.

1.3 Notation

We define types and contexts for Gödel's System T using a standard BNF grammar in Figure 1. However, we use intrinsically typed terms instead of defining a grammar for terms and an extrinsic typing relation. Below is an example of the standard extrinsic typing of expressions.

$$A, B, C ::= \mathbb{N} \mid A \supset B$$
$$\Gamma, \Delta, \Xi, \Phi ::= \emptyset \mid \Gamma, A$$

Fig. 1: BNF grammar for *types* and *contexts*. Expressions, values, and neutrals are not represented by a BNF grammar. Instead, they are defined by an intrinsically typed representation, rather than a BNF grammar and an extrinsic typing relation over said grammar.

$$e ::= ... \mid \text{zero} \mid \text{suc } n \mid ...$$

$$\frac{\Gamma \vdash^{\text{E}} n : \mathbb{N}}{\Gamma \vdash^{\text{E}} \text{suc } n : \mathbb{N}}\ \mathbb{N}\text{-I}_2$$

Instead, we use the proof terms of our typing relation as intrinsically typed terms (combining both definitions above into the single definition below).

$$\frac{(n : \Gamma \vdash^{\text{E}} \mathbb{N})}{\Gamma \vdash^{\text{E}} \mathbb{N}}\ \text{suc } n$$

We label each premise with a variable to the left of a colon, and the name of the typing rule shows how to use it as a proof term by applying it to the premise labels. In other words, the labels represent the derivations of the premises.

The letter appearing as the superscript to the turnstyle is part of the name of each intrinsic typing judgment. For example, $\vdash^{\text{E}}$ types expressions in Figure 3, $\vdash^{\text{V}}$ types values in Figure 4, and $\vdash^{\text{N}}$ types neutrals in Figure 5.

Finally, we have taken care to ensure that all of our constructions are easily formalizable. For this reason, we use De Bruijn variables [8]. For example, our typing rule for functions does not explicitly bind its variable.

$$\frac{(b : \Gamma, A \vdash^{\text{E}} B)}{\Gamma \vdash^{\text{E}} A \supset B}\ \lambda b$$

Additionally, proof that a (De Bruijn) variable of a particular type exists in the context is given by the judgment $\vdash^{\text{R}}$ in Figure 2. Mentioning a variable in a term thus requires evidence that the variable judgment is satisfied.

$$\frac{(i : \Gamma \vdash^{\text{R}} A)}{\Gamma \vdash^{\text{E}} A}\ \text{var } i$$

$$\frac{}{\Gamma, A \vdash^{\mathrm{R}} A}\ \text{here} \qquad \frac{(i : \Gamma \vdash^{\mathrm{R}} A)}{\Gamma, B \vdash^{\mathrm{R}} A}\ \text{there}\ i$$

Fig. 2: Intrinsic typing of De Bruijn *variables*. The intrinsically typed variables act as proofs that the type parameter of the judgment appears in the context parameter of the judgment.

$$\frac{}{\Gamma \vdash^{\mathrm{E}} \mathbb{N}}\ \text{zero} \qquad \frac{(n : \Gamma \vdash^{\mathrm{E}} \mathbb{N})}{\Gamma \vdash^{\mathrm{E}} \mathbb{N}}\ \text{suc}\ n$$

$$\frac{(b : \Gamma, A \vdash^{\mathrm{E}} B)}{\Gamma \vdash^{\mathrm{E}} A \supset B}\ \lambda b \qquad \frac{(i : \Gamma \vdash^{\mathrm{R}} A)}{\Gamma \vdash^{\mathrm{E}} A}\ \text{var}\ i$$

$$\frac{(f : \Gamma \vdash^{\mathrm{E}} A \supset B) \quad (a : \Gamma \vdash^{\mathrm{E}} A)}{\Gamma \vdash^{\mathrm{E}} B}\ f \cdot a$$

$$\frac{(n : \Gamma \vdash^{\mathrm{E}} \mathbb{N}) \quad (c_z : \Gamma \vdash^{\mathrm{E}} C) \quad (c_s : \Gamma \vdash^{\mathrm{E}} C \supset C)}{\Gamma \vdash^{\mathrm{E}} C}\ \text{rec}\ n\ c_z\ c_s$$

Fig. 3: Intrinsic typing of *expressions* for Gödel's System T. The intrinsically typed expressions act as proofs that the expression represented by the proof term is well typed.

2 Overview of NbE and SbE

First we review why the proofs of termination of naive definitions of normalization and hereditary substitution fail, and then we provide a high-level overview of theorems and definitions necessary to define NbE and SbE (which intrinsically terminate, unlike the naive definitions). Finally, we discuss the similarities and differences between NbE and SbE. Henceforth, the development will proceed in a top-down manner. Each theorem comes in four parts:

1. The statement of the theorem.
2. A table of lemmas that the theorem depends on in its proof, and references to where proofs of the lemmas can be found.
3. A high-level discussion of the theorem and its proof.
4. The low-level details of the proof, sometimes interspersed with high-level explanations of how it works.

Preexisting proofs that we reuse from the NbE literature can be found in the appendices, while novel proofs appear in the body of the paper.

$$\frac{}{\Gamma \vdash^{\mathrm{V}} \mathbb{N}}\ \text{zero} \qquad \frac{(n : \Gamma \vdash^{\mathrm{V}} \mathbb{N})}{\Gamma \vdash^{\mathrm{V}} \mathbb{N}}\ \text{suc } n$$

$$\frac{(b : \Gamma, A \vdash^{\mathrm{V}} B)}{\Gamma \vdash^{\mathrm{V}} A \supset B}\ \lambda b \qquad \frac{(a : \Gamma \vdash^{\mathrm{N}} A)}{\Gamma \vdash^{\mathrm{V}} A}\ \text{neut } a$$

Fig. 4: Intrinsic typing of *values* (canonical terms) for Gödel's System T. The intrinsically typed values act as proofs that the value represented by the proof term is well typed. The grammar of values also includes all neutral terms (see Figure 5).

2.1 Termination Issues with Naive Definitions

When defining normalization and hereditary substitution, the main problem preventing an obvious termination argument by structural induction is the lack of an inductive hypothesis in the function application case.

Naive Normalization Consider the application case in the definition of strong normalization below, given as a big-step binary relation between expressions. Below "here" refers to De Bruijn index zero from Figure 2.

$$\frac{f \Downarrow \lambda b \quad a \Downarrow a' \quad [a'/\text{here}]b \Downarrow b'}{f \cdot a \Downarrow b'}$$

After evaluating the function to a lambda, we need to substitute the argument into the lambda body. This may introduce new β-redexes, so we must evaluate again but there is no obvious termination measure to justify this. Above, we have already appealed to the inductive hypothesis for the two subterms f (the function) and a (its argument), and there is no inductive hypothesis for the function λ-body b.

Naive Hereditary Substitution Consider the application case in the definition of hereditary substitution below, given as a ternary relation between an input value, a mapping from variables to values (σ), and a resulting value. Note that we actually have two mutually defined relations, one for substituting into a value and producing a new value ($[\sigma]v =^{\mathrm{V}} v'$), and another for substituting into neutral but also producing a value ($[\sigma]n =^{\mathrm{N}} v$).

$$\frac{[\sigma]f =^{\mathrm{N}} \lambda b \quad [\sigma]a =^{\mathrm{V}} a' \quad [\sigma, a']b =^{\mathrm{V}} b'}{[\sigma](f \cdot a) =^{\mathrm{N}} b'}$$

This relation is not obviously terminating for the same reason as normalization. We have inductive hypotheses for substituting into the function and substituting into the argument, but there is no inductive hypothesis for substituting into the λ-body.

$$\frac{(i : \Gamma \vdash^{\mathrm{R}} A)}{\Gamma \vdash^{\mathrm{N}} A}\ \mathrm{var}\ i$$

$$\frac{(f : \Gamma \vdash^{\mathrm{N}} A \supset B) \quad (a : \Gamma \vdash^{\mathrm{V}} A)}{\Gamma \vdash^{\mathrm{N}} B}\ f \cdot a$$

$$\frac{(n : \Gamma \vdash^{\mathrm{N}} \mathbb{N}) \quad (c_z : \Gamma \vdash^{\mathrm{V}} C) \quad (c_s : \Gamma \vdash^{\mathrm{V}} C \supset C)}{\Gamma \vdash^{\mathrm{N}} C}\ \mathrm{rec}\ n\ c_z\ c_s$$

Fig. 5: Intrinsic typing of *neutrals* (variables and elimination rules) for Gödel's System T. The intrinsically typed neutrals act as proofs that the neutral represented by the proof term is well typed.

2.2 The Model

The naive definition of normalization is defined in one phase, going directly from syntactic expressions to syntactic values. In contrast, NbE is defined in two phases. The first (called "evaluation") phase goes from syntactic expressions to semantic values (of a Kripke model). The second phase (called "reification") goes from semantic values to syntactic values. In the naive definition, normalization (producing a syntactic value) is a large definition given by case analysis over all syntactic forms of expressions. In NbE, evaluation (producing a semantic value) is a large definition given by case analysis over all syntactic forms of expressions, while reification is a small definition given only by case analysis over the types of semantic values.

NbE breaks up normalization into two phases (evaluation into the model and reification back out of it) so that both phases can be defined in a structurally terminating way. There are no termination issues when defining reification (the smaller phase). However, evaluation (the larger phase) must be defined by case analysis over all expressions. Recall that the problematic case in the definition of normalization is function application. The definition of the model below makes it possible to define the function application case of evaluation in a structurally terminating way.

Definition 1 (Semantic Values).
The model of values $[\![\ \Gamma \vdash A\ ]\!]$ *denotes a set (a type in the metalanguage), and it is defined inductively on the structure of (object language) types (A). We refer to any member of the model of values as a "semantic value".*

Case (Natural numbers).

$$[\![\ \Gamma \vdash \mathbb{N}\ ]\!] \triangleq \Gamma \vdash^{V} \mathbb{N}$$

Syntactic natural number values are also semantic natural numbers values.

Case (Functions).

$$[\![\ \Gamma \vdash A \supset B\ ]\!] \triangleq \forall \Delta.\ [\![\ \Gamma, \Delta \vdash A\ ]\!] \to [\![\ \Gamma, \Delta \vdash B\ ]\!]$$

The model of functions is a function in the metalanguage. The semantic function takes a weakened semantic value (whose type is the domain of the function) as an argument, and produces a weakened semantic value (whose type is the codomain of the function). Both the input and output values of the model function are weakened by some arbitrary context Δ, which can be empty. [2]

So how does this model help us to define the application case of evaluation? The result of evaluation (rather than normalization) is a semantic value (rather than a syntactic value). Thus, evaluating the function produces a semantic function (a function in the metalanguage), which we can apply to the semantic value produced by evaluating the argument (in this case the weakening Δ is merely the empty context). Hence, we can define the application case of evaluation in an structurally terminating way since it only needs to appeal to the inductive hypotheses for the function and its argument. The inductive hypothesis for the function produces a metalanguage function that we can apply, rather than a syntactic λ and a body for which we would be missing an inductive hypothesis.

Given that evaluation must map syntactic expressions to semantic values, what should it do when it encounters a variable? We can make evaluation take an additional argument, called the "semantic environment", mapping variables to semantic values. More specifically, the semantic environment maps all types (where each type represents a variable) in some initial context Γ to semantic values in some terminal context Δ. Because the environment values are scoped by Δ, evaluation works on open terms, hence NbE can define strong normalization.

Definition 2 (Semantic Environments).
The model of environments $[\![\ \Delta \vdash \Gamma\ ]\!]$ is defined inductively on the structure of the second argument (Γ). We refer to any member of the model of environments as a "semantic environment".

Case (Empty context).

$$[\![\ \Delta \vdash \emptyset\ ]\!] \triangleq \top$$

The empty context is trivially inhabited in the environment model.

Case (Context extension).

$$[\![\ \Delta \vdash \Gamma, A\ ]\!] \triangleq [\![\ \Delta \vdash \Gamma\ ]\!] \times [\![\ \Delta \vdash A\ ]\!]$$

The environment model of a context extension consists of two parts, given as a pair type. The first part is a semantic environment for the context being extended (Γ). The second part is a semantic value in the terminal context (Δ), whose type is the type of the context extension (A).

[2] In a true Kripke model the function case includes an accessibility relation between contexts (or "worlds"), allowing the semantic function domain and codomain to be scoped under an arbitrary context Δ. Merely out of preference, our "Kripke-like" model restricts the domain and codomain to be a weakening (Γ, Δ) of the original context (Γ).

2.3 Normalization by Evaluation

So far we have seen the model of values and environments, and the intuition behind the model enabling a terminating definition of evaluation. Now let's take a closer look at how exactly to define normalization in terms of evaluation. What theorems [3] do we need, and how do we instantiate these theorems in the proof of normalization?

Theorem 1 (Normalization by Evaluation).

$$\mathrm{nbe} : \Gamma \vdash^{\mathrm{E}} A \to \Gamma \vdash^{\mathrm{V}} A$$

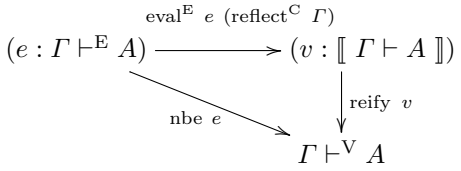
Appendix A	Theorem 5	$\mathrm{eval}^{\mathrm{E}}$	$\Gamma \vdash^{\mathrm{E}} A \to [\![\ \Delta \vdash \Gamma\ ]\!] \to [\![\ \Delta \vdash A\ ]\!]$
Appendix B	Theorem 6	reify	$[\![\ \Gamma \vdash A\ ]\!] \to \Gamma \vdash^{\mathrm{V}} A$
Appendix C	Theorem 7	$\mathrm{reflect}^{\mathrm{C}}$	$\forall \Gamma.\ [\![\ \Gamma \vdash \Gamma\ ]\!]$

The high-level plan of NbE is to define normalization as the composition of evaluation ($\mathrm{eval}^{\mathrm{E}}$) of a syntactic expression to a semantic value, and reification (reify) back to a syntactic value. Normalization takes a syntactic expression to a syntactic value, while evaluation takes a syntactic expression *and* a semantic environment to a semantic value. So what should we use for the semantic environment argument when defining normalization in terms of evaluation? We can reflect any context (Γ) as a semantic environment ($\mathrm{reflect}^{\mathrm{C}}$). Specifically, a context is reflected as the identity semantic environment, mapping syntactic variables to themselves, but as semantic values.

Proof.

$$\begin{aligned}
e &: \Gamma \vdash^{\mathrm{E}} A && (\textit{assumption})\\
\sigma &: [\![\ \Gamma \vdash \Gamma\ ]\!] && (\mathrm{reflect}^{\mathrm{C}}\ \Gamma)\\
v &: [\![\ \Gamma \vdash A\ ]\!] && (\mathrm{eval}^{\mathrm{E}}\ e\ \sigma)\\
v' &: \Gamma \vdash^{\mathrm{V}} A && (\mathrm{reify}\ v)
\end{aligned}$$

The diagram of NbE in the introduction left out some details, as it was only meant to convey the intuition behind NbE. However, we can fill in the details to arrive at the alternative pictorial proof below.



□

[3] Because the "definition" of normalization contains an intrinsic termination proof, we refer to normalization, evaluation, and related "definitions" as *theorems* and *lemmas*.

2.4 Hereditary Substitution by Canonical Evaluation

How is the type of hereditary substitution different from that of normalization? First, we are substituting into a *value* rather than normalizing an *expression*. Second, hereditary substitution also takes a *syntactic* environment, a mapping of variables to syntactic values, as an additional argument. [4]

Definition 3 (Syntactic Environments).
Syntactic environments $\Delta \vdash^{V} \Gamma$ are defined as a cartesian product, exactly like semantic environments (Definition 2, $[\![\ \Delta \vdash \Gamma\ ]\!]$), except they contain syntactic values instead of semantic values.

Theorem 2 (Hereditary Substitution by Canonical Evaluation).

$$\text{sbe} : \Gamma \vdash^{\text{V}} A \to \Delta \vdash^{\text{V}} \Gamma \to \Delta \vdash^{\text{V}} A$$

Section 3	Theorem 3	eval^{V}	$\Gamma \vdash^{\text{V}} A \to [\![\ \Delta \vdash \Gamma\ ]\!] \to [\![\ \Delta \vdash A\ ]\!]$
Appendix B	Theorem 6	reify	$[\![\ \Gamma \vdash A\ ]\!] \to \Gamma \vdash^{\text{V}} A$
Section 4	Theorem 4	$\text{reflect}^{\text{S}}$	$\forall \Delta.\ \Delta \vdash^{\text{V}} \Gamma \to [\![\ \Delta \vdash \Gamma\ ]\!]$

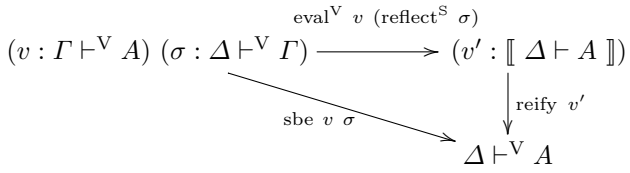
Above are the theorems we need for SbE. In particular, notice that we get to reuse reify from NbE. The high-level plan of SbE is to define hereditary substitution as the composition of canonical evaluation (eval^{V}) of a syntactic value to a semantic value, and reification (reify) back to a syntactic value. What exactly is canonical evaluation? Recall that evaluation of expressions turns syntactic expressions into semantic values, reducing β-redexes and replacing variables with their definition in the semantic environment (σ) along the way. On the other hand, canonical evaluation takes a syntactic value and a semantic environment to a semantic value. It will not come across β-redexes in the input syntactic value, but replacing variables with semantic values (from σ) may *introduce* β-redexes that need to be immediately reduced. Finally, the syntactic environment argument (σ) of SbE is reflected ($\text{reflect}^{\text{S}}$) as a semantic environment, and is used as the semantic environment argument of canonical evaluation.

Proof.

$$\begin{aligned}
v &: \Gamma \vdash^{\text{V}} A && (\textit{assumption}) \\
\sigma &: \Delta \vdash^{\text{V}} \Gamma && (\textit{assumption}) \\
\sigma' &: [\![\ \Delta \vdash \Gamma\ ]\!] && (\text{reflect}^{\text{S}}\ \sigma) \\
v' &: [\![\ \Delta \vdash A\ ]\!] && (\text{eval}^{\text{V}}\ v\ \sigma') \\
v'' &: \Delta \vdash^{\text{V}} A && (\text{reify}\ A\ v')
\end{aligned}$$

[4] What we call a "syntactic environment" is normally called a "substitution". Because we refer so much to the operation of hereditary substitution, it is less ambiguous to refer to its "substitution" argument as a "syntactic environment".

Once again, we can add detail to the commutative diagram of SbE in the introduction to arrive at the alternative pictorial proof below.



□

2.5 NbE versus SbE

In the world of expressions, evaluation takes an additional argument (the semantic environment) compared to normalization, which only takes the expression to reduce. But, in the world of values hereditary substitution and canonical evaluation both take a value, an environment, and produce a value. The only difference is that canonical evaluation takes a semantic environment and produces a semantic value. In other words, canonical evaluation *is* a model-theoretic hereditary substitution, compared to the typical proof-theoretic hereditary substitution. What a beautiful coincidence!

A consequence of this is that normalization must invent a semantic environment to pass to evaluation, because it does not already have a syntactic environment argument to reflect. Because normalization does not change the value of free variables, the identity semantic environment can be used by reflecting the context (reflect$^{\mathrm{C}}$). In contrast, hereditary substitution already has a syntactic environment (that is used to look up the value of any free variables), so canonical evaluation takes the reflection (reflect$^{\mathrm{S}}$) of the syntactic environment as its semantic environment argument. More generally, an important difference between normalization and hereditary substitution is that the former preserves the type (A) and context (Γ) of its input, while the latter only preserves its type (the context changes to Δ).

Although there is a more direct correspondence between SbE and canonical evaluation (than there is between NbE and evaluation), there is nothing preventing us from defining a function on expressions that normalizes β-redexes, and simultaneously performs hereditary substitutions from an environment mapping variables to expressions. This normalization plus hereditary substitution function (nsbe) would take an extra syntactic expression environment argument, creating a correspondence with evaluation that is just as direct as the one that SbE enjoys.

$$\mathrm{nsbe} : \Gamma \vdash^{\mathrm{E}} A \to \Delta \vdash^{\mathrm{E}} \Gamma \to \Delta \vdash^{\mathrm{V}} A$$

3 Canonical Evaluation

Canonical evaluation (eval^{V}) is the primary definition in SbE, taking a syntactic value and a semantic environment to a semantic value. It is analogous to evaluation (eval^{E}) in NbE. In fact canonical evaluation is defined exactly the same way that evaluation is, except by case analysis over values instead of expressions!

Because the grammar of values is mutually defined with neutrals, canonical evaluation of values (eval^{V}) is mutually defined with canonical evaluation of neutrals (eval^{N}). Recall that values consist of axioms and inference rules for all constructors, plus an inference rule for injecting neutrals into values. A neutral is a variable, or sequence of eliminations (i.e. applications or primitive recursions) that begins with the elimination of a variable.

Expressions and values are quite different, as the former may contain β-redexes. However, if you compare canonical evaluation (Theorem 3 & Lemma 1) with evaluation (Theorem 5 in Appendix A) you will notice that they are identical, modulo the former being defined over a split grammar. Why is that? An expression may already be a syntactic β-redex, while a syntactic value will never be a syntactic β-redex. But, both an expression and a value may become a semantic β-redex (a β-redex of the metalanguage) after replacing syntactic variables with values from the semantic environment.

Essentially, evaluation and canonical evaluation are similar because they operate on variables the same way, replacing them with semantic values from the semantic environment. In contrast, normalization leaves variables unchanged, while hereditary substitution replaces variables with syntactic values from the syntactic environment.

Theorem 3 (Canonical Evaluation of Values).

$$\text{eval}^{\text{V}} : \Gamma \vdash^{\text{V}} A \to [\![\ \Delta \vdash \Gamma\ ]\!] \to [\![\ \Delta \vdash A\ ]\!]$$

Section 3	Lemma 1	eval^{N}	$\Gamma \vdash^{\text{N}} A \to [\![\ \Delta \vdash \Gamma\ ]\!] \to [\![\ \Delta \vdash A\ ]\!]$
Appendix C	Lemma 4	mono^{S}	$\forall \Delta.\ [\![\ \Xi \vdash \Gamma\ ]\!] \to [\![\ \Xi, \Delta \vdash \Gamma\ ]\!]$

Canonical evaluation (eval^{V}) is a type-preserving translation. It takes a syntactic value in some initial context ($\Gamma \vdash^{\text{V}} A$) and a semantic environment in some terminal context ($[\![\ \Delta \vdash \Gamma\ ]\!]$), and produces a semantic value in the terminal context ($[\![\ \Delta \vdash A\ ]\!]$). The semantic environment has a semantic value for every type in the initial context, each of which must be scoped in the terminal context.

Proof. By induction on the value $\Gamma \vdash^{V} A$.

Case (Zero). The zero case is immediate.

$$\begin{array}{ll} n : \Delta \vdash^{\text{V}} \mathbb{N} & (\text{zero}) \\ n : [\![\ \Delta \vdash \mathbb{N}\ ]\!] & (\textit{definition}) \end{array}$$

Case (Successor). The successor case is a congruence.

$$
\begin{array}{ll}
n : \Gamma \vdash^{\mathrm{V}} \mathbb{N} & (\textit{assumption}) \\
\sigma : [\![\ \Delta \vdash \Gamma\ ]\!] & (\textit{assumption}) \\
n' : [\![\ \Delta \vdash \mathbb{N}\ ]\!] & (\textit{i.h.}\ \mathrm{eval}^{\mathrm{V}}\ n\ \sigma) \\
n' : \Delta \vdash^{\mathrm{V}} \mathbb{N} & (\textit{definition}) \\
m : \Delta \vdash^{\mathrm{V}} \mathbb{N} & (\mathrm{suc}\ n') \\
m : [\![\ \Delta \vdash \mathbb{N}\ ]\!] & (\textit{definition})
\end{array}
$$

Case (Function). When canonically evaluating a function (a λ term), the return type is a model function (a metalanguage function). We get two standard parameters from the theorem (b and σ), but because we return a metalanguage function we also get two additional parameters from its domain (Ξ and a). The Ξ parameter is a context to weaken the result with, and the a parameter is a semantic value that has been weakened by Ξ. Review the function case of $[\![\ \Gamma \vdash A \supset B\ ]\!]$ in Definition 1 to see where the two additional parameters come from.

We would like to produce a result by canonically evaluating the λ-body b, using the semantic environment σ extended by the extra argument a that we received from the model. However, all of the semantic values in the semantic environment are scoped under Δ, while semantic value a is scoped under Δ, Ξ. Thus, we must use the $\mathrm{mono}^{\mathrm{S}}$ lemma from Appendix C to transform σ. The reader can recognize $\mathrm{mono}^{\mathrm{S}}$ as a form of weakening (we are weakening Δ by Ξ above) lifted to semantic environments.

$$
\begin{array}{ll}
b : \Gamma, A \vdash^{\mathrm{V}} B & (\textit{assumption}) \\
\sigma : [\![\ \Delta \vdash \Gamma\ ]\!] & (\textit{assumption}) \\
\Xi & (\textit{assumption}) \\
a : [\![\Delta, \Xi \vdash A]\!] & (\textit{assumption}) \\
\sigma' : [\![\ \Delta, \Xi \vdash \Gamma\ ]\!] & (\mathrm{mono}^{\mathrm{S}}\ \Xi\ \sigma) \\
b' : [\![\ \Delta \vdash B\ ]\!] & (\textit{i.h.}\ \mathrm{eval}^{\mathrm{V}}\ b\ (\sigma', a))
\end{array}
$$

Case (Neutral). Canonical evaluation of neutrals is delegated to a mutually defined theorem ($\mathrm{eval}^{\mathrm{N}}$).

$$
\begin{array}{ll}
a : \Gamma \vdash^{\mathrm{N}} A & (\textit{assumption}) \\
\sigma : [\![\ \Delta \vdash \Gamma\ ]\!] & (\textit{assumption}) \\
a' : [\![\ \Delta \vdash A\ ]\!] & (\textit{i.h.}\ \mathrm{eval}^{\mathrm{N}}\ a\ \sigma)
\end{array}
$$

□

Lemma 1 (Canonical Evaluation of Neutrals).

$$\text{eval}^{\text{N}} : \Gamma \vdash^{\text{N}} A \to [\![\ \Delta \vdash \Gamma\ ]\!] \to [\![\ \Delta \vdash A\ ]\!]$$

Section 3	Theorem 3	eval^{V}	$\Gamma \vdash^{\text{V}} A \to [\![\ \Delta \vdash \Gamma\ ]\!] \to [\![\ \Delta \vdash A\ ]\!]$
Appendix A	Lemma 2	$[\![\text{rec}]\!]$	$[\![\ \Gamma \vdash C\ ]\!] \to [\![\ \Gamma \vdash C \supset C\ ]\!] \to [\![\ \Gamma \vdash \mathbb{N}\ ]\!] \to [\![\ \Gamma \vdash C\ ]\!]$

Canonical evaluation of neutrals (eval^{N}) handles the canonical evaluation of variables and eliminations, and is used as a lemma in the mutually defined canonical evaluation of values (eval^{V}).

Proof. *By induction on the neutral* $\Gamma \vdash^{N} A$.

Case (Variable). Canonically evaluate a variable by looking up its corresponding semantic value in the semantic environment.

$$\begin{aligned} i &: \Gamma \vdash^{\text{R}} A && (\textit{assumption}) \\ \sigma &: [\![\ \Delta \vdash \Gamma\ ]\!] && (\textit{assumption}) \\ a &: [\![\ \Delta \vdash A\ ]\!] && (\textit{lookup}\ \sigma[i]) \end{aligned}$$

Case (Application). The whole point of the value model (Definition 1) is to make this case easy to define. Evaluating the applied function (the first inductive hypothesis) results in a metalanguage function, which we can apply to an empty context extension ($\emptyset$) and the result of evaluating the argument of the function application (the second inductive hypothesis).

$$\begin{aligned} f &: \Gamma \vdash^{\text{N}} A \supset B && (\textit{assumption}) \\ a &: \Gamma \vdash^{\text{V}} A && (\textit{assumption}) \\ \sigma &: [\![\ \Delta \vdash \Gamma\ ]\!] && (\textit{assumption}) \\ f' &: [\![\ \Delta \vdash A \supset B\ ]\!] && (\textit{i.h.}\ \text{eval}^{\text{N}}\ f\ \sigma) \\ f' &: \forall \Delta.\ [\![\ \Gamma, \Delta \vdash A\ ]\!] \to [\![\ \Gamma, \Delta \vdash B\ ]\!] && (\textit{definition}) \\ a' &: [\![\ \Delta \vdash A\ ]\!] && (\textit{i.h.}\ \text{eval}^{\text{V}}\ a\ \sigma) \\ b &: [\![\ \Delta \vdash B\ ]\!] && (f'\ \emptyset\ a') \end{aligned}$$

Case (Primitive recursion). When canonically evaluating a neutral primitive recursion (rec), we can get inductive hypotheses for all the arguments to the primitive recursion, but we must perform a semantic version of primitive recursion using a lemma ($[\![\text{rec}]\!]$). The lemma $[\![\text{rec}]\!]$ was developed for evaluation

in NbE, and is given in Appendix A as Lemma 2.

$$
\begin{array}{rl}
c_z : \Gamma \vdash^{\mathrm{V}} C & (\textit{assumption}) \\
c_s : \Gamma \vdash^{\mathrm{V}} C \supset C & (\textit{assumption}) \\
n : \Gamma \vdash^{\mathrm{N}} \mathbb{N} & (\textit{assumption}) \\
\sigma : [\![\ \Delta \vdash \Gamma\ ]\!] & (\textit{assumption}) \\
c'_z : [\![\ \Delta \vdash C\ ]\!] & (\textit{i.h. } \mathrm{eval}^{\mathrm{V}}\ c_z\ \sigma) \\
c'_s : [\![\ \Delta \vdash C \supset C\ ]\!] & (\textit{i.h. } \mathrm{eval}^{\mathrm{V}}\ c_s\ \sigma) \\
n' : [\![\ \Delta \vdash \mathbb{N}\ ]\!] & (\textit{i.h. } \mathrm{eval}^{\mathrm{N}}\ n\ \sigma) \\
c : [\![\ \Delta \vdash C\ ]\!] & ([\![\mathrm{rec}]\!]\ c'_z\ c'_s\ n')
\end{array}
$$

□

4 Environment Reflection

Environment reflection takes a context (Δ) and a syntactic environment ($\Delta \vdash^{\mathrm{V}} \Gamma$), and produces a semantic environment ($[\![\ \Delta \vdash \Gamma\ ]\!]$). Recall that a syntactic environment (Definition 3 in Section 2) is a tuple of syntactic values, all scoped under the same context. In contrast, a semantic environment (Definition 2 in Section 2) is a tuple of semantic values, all scoped under the same context.

Theorem 4 (Environment Reflection).

$$\mathrm{reflect}^{\mathrm{S}} : \forall \Delta.\ \Delta \vdash^{\mathrm{V}} \Gamma \to [\![\ \Delta \vdash \Gamma\ ]\!]$$

Section 3	Theorem 3	$\mathrm{eval}^{\mathrm{V}}$	$\Gamma \vdash^{\mathrm{V}} A \to [\![\ \Delta \vdash \Gamma\ ]\!] \to [\![\ \Delta \vdash A\ ]\!]$
Appendix C	Theorem 7	$\mathrm{reflect}^{\mathrm{C}}$	$\forall \Gamma.\ [\![\ \Gamma \vdash \Gamma\ ]\!]$

Environment reflection is a context-preserving mapping from syntactic environments to semantic environments. So how do we translate a single syntactic value ($\Delta \vdash^{\mathrm{V}} A$) in the syntactic environment ($\Delta \vdash^{\mathrm{V}} \Gamma$) to a semantic value ($[\![\ \Delta \vdash A\ ]\!]$) in the same context? We canonically evaluate it, but what should the semantic environment for canonical evaluation be? Because we do not want the context to change in the result of canonical evaluation, we can pass it the identity semantic environment ($[\![\ \Delta \vdash \Delta\ ]\!]$) via context reflection (Theorem 7 in Appendix C).

Proof. By induction on the environment $\Delta \vdash^{V} \Gamma$.

Case (Empty environment). The empty environment case is trivial.

$$
\begin{array}{ll}
\Delta & (\textit{assumption}) \\
u : \top & (\textit{assumption}) \\
u : [\![\ \Delta \vdash \Gamma\ ]\!] & (\textit{definition})
\end{array}
$$

Case (Environment extension). To reflect a syntactic environment (σ) extended by a syntactic value (x), we must produce a semantic environment extended by a semantic value. The induction hypothesis gives us the semantic environment.

To get the semantic value we must canonically evaluate the syntactic value argument (x). The second parameter of canonical evaluation is an environment, which determines the context that the resulting semantic value is scoped under. Because our syntactic value x already has the correct scope (Δ), we can use context reflection (of Δ) as the semantic environment argument for canonical evaluation.

$$
\begin{array}{rl}
\Delta & (\textit{assumption}) \\
\sigma : \Delta \vdash^{\mathrm{V}} \Gamma & (\textit{assumption}) \\
x : \Delta \vdash^{\mathrm{V}} A & (\textit{assumption}) \\
\sigma' : [\![\ \Delta \vdash \Gamma\ ]\!] & (\textit{i.h.}\ \mathrm{reflect}^{\mathrm{S}}\ \Delta\ \sigma) \\
\delta : [\![\ \Delta \vdash \Delta\ ]\!] & (\mathrm{reflect}^{\mathrm{C}}\ \Delta) \\
x' : [\![\ \Delta \vdash A\ ]\!] & (\mathrm{eval}^{\mathrm{V}}\ x\ \delta) \\
\sigma'' : [\![\ \Delta \vdash \Gamma, A\ ]\!] & (\sigma', x')
\end{array}
$$

□

4.1 No Circularity

If you look back at the definition of SbE (Theorem 2 in Section 2), it might now seem circular. SbE is defined by canonically evaluating a syntactic value, but canonical evaluation also takes a semantic environment. So before canonically evaluating the syntactic value, we must reflect the syntactic environment to a semantic environment. However, environment reflection (Theorem 4) is once again defined by canonical evaluation, which needs a semantic environment for each syntactic value in the reflected syntactic environment.

Fortunately, the circularity is broken because we can always produce an identity semantic environment (mapping variables to themselves as semantic values) out of thin air using context reflection (Theorem 7 in Appendix C).

5 Related & Future Work

Below we compare and contrast model-theoretic hereditary substitution (SbE) with conventional proof-theoretic hereditary substitution. In particular, we point out that the model-theoretic version scales to calculi with inductive types. We also discuss some future work, namely scaling SbE to calculi with features like delimited control, and proving correctness of SbE.

5.1 Proof-Theoretic Hereditary Substitution

Techniques similar to hereditary substitution originated in the world of proof theory. For example, Prawitz [24] proved that normalization of natural deduction terminates via a lexicographic ordering on proposition (types) and proofs (terms). This syntactic proof is similar to a definition of hereditary substitution. Joachimski et al. [13] showed that normalization of various expression-based λ-calculi (even Gödel's System T) terminates using a syntactic argument also resembling hereditary substitution. Watkins [26] explicitly defined hereditary substitution by proving termination of a semantics for a concurrent logical framework.

There has also been work on formalizing the syntactic termination argument of hereditary substitution in type theory. This is done by defining hereditary substitution as a total function with an implicit termination proof that relies on the lexicographic termination argument on types and terms. For example, Keller and Altenkirch [14] define hereditary substitution this way within Agda for the Simply Typed Lambda Calculus (STLC). Similarly, Abel [1] defines simultaneous hereditary substitution for STLC in Agda. However, extending formal, syntactic, and implicit termination arguments of hereditary substitution to richer calculi has been more difficult. For example, Eades [9] discusses the difficulties with extending proof-theoretic hereditary substitution to richer calculi. In particular, the termination argument becomes troublesome in calculi with inductive types. Eades [10] has overcome the problems for predicative System F and STLC with sum types. However, formal and implicit termination arguments of hereditary substitution for calculi with recursive sum types, such as natural numbers (Gödel's System T) or parametric lists, remain to be discovered.

5.2 Model-Theoretic Hereditary Substitution

SbE is a model-theoretic definition of hereditary substitution. It is derived from NbE by adapting evaluation to work on canonical terms, rather than expressions. Because SbE reuses so much from NbE, it also enjoys many of its benefits. For example, NbE can be used to define normalization of STLC within type theory as a function with an implicit termination proof. Coquand [6] first formalized NbE for STLC in Alf [16]. It is also easy to formalize NbE for STLC in Agda, as demonstrated by Morrison [20] and Ilik [12].

Significantly, type theoretic formalizations of NbE have been easy to extend to more complex calculi, such as calculi with inductive types. Ilik [12] shows how to formalize NbE in Agda for a λ-calculus with sums, polymorphic lists, and even delimited control.

Because SbE is an adaption of NbE to canonical terms, it is easy to adapt formalizations of NbE for complex calculi to SbE. For example, our accompanying code implements SbE for a calculus with polymorphic lists and folds over them. It should be just as easy to adapt Ilik's NbE formalization of delimited control to SbE.

5.3 Correctness

Because NbE allows you to write normalization as a function within type theory, you may be content with this normalization function acting as both the specification and the implementation of the semantics for your calculus. Otherwise, you may wish to prove that normalization is both sound and complete with respect to a relationally specified equivalence relation on expressions.

Similarly, you may accept normalization defined in terms of SbE (nbs in Appendix D) as the specification and implementation of a semantics for expressions. If you do not accept this, then you may wish to prove the following correctness theorem.

$$\text{nbs } e_1 \equiv \text{nbs } e_2 \;\leftrightarrow\; e_1 \approx_\beta e_2$$

The theorem requires normalization by hereditary substitution (nbs) to be sound (the "if" direction) and complete (the "only if" direction) with respect to a relational specified β-equivalence relation. Note that because we are using De Bruijn variables, the left-hand side only requires strict equality between the values produced from normalizing the input expressions.

We have not yet proven the theorem above for SbE, but because so much of SbE is derived from NbE, we expect proving this theorem to borrow proof machinery from the NbE world. We expect the completeness direction to be proven using a logical relation, similar to the proof that Abel [2] gives for NbE completeness. We expect the soundness direction of SbE to be provable directly, like the soundness direction of NbE also mentioned by Abel [2]. We expect the soundness proof to require several "commutation lemmas", as described by Keller and Altenkirch [14], and as used in their correctness proof of syntactic hereditary substitution. The commutation lemmas ensure that hereditary substitution into values commutes with ordinary substitution into expressions.

6 Conclusion

Normalization by Evaluation (NbE) makes it possible to formalize an intrinsic termination argument of the normalization of expressions for many λ-calculi (e.g. calculi with polymorphic lists, sums, continuations, etc).

If you adapt the evaluation part of NbE from expressions to canonical terms, you get a function with an intrinsic termination proof that maps a syntactic value and a semantic environment to a semantic value. We call this function canonical evaluation. By beautiful coincidence, canonical evaluation is *semantic* hereditary substitution. In other words, it is the lifting of *syntactic* hereditary substitution to semantic values of a Kripke model.

After noticing that canonical evaluation is semantic hereditary substitution, we showed how to define syntactic hereditary substitution by reflecting its environment argument and reifying the result of canonical evaluation. We call this technique Hereditary Substitution by Canonical Evaluation (SbE). When defining SbE, we get to reuse many definitions, theorems, and lemmas from NbE,

such as the model and reification. Finally, hereditary substitution for complex λ-calculi can be formalized within dependently typed languages as a function with an intrinsic termination proof!

Acknowledgments. We would like to thank Andrew Cave for helping us understand subtle issues related to context weakening when formalizing termination proofs via realizability predicates. We would also like to thank Darin Morrison for originally introducing us to the connection between NbE and a Kripke model for intuitionistic logic. Finally, we would like to thank Clarissa Littler and Ki Yung Ahn for giving valuable feedback on drafts of this paper. This work was supported by NSF/CISE/CCF grant #1320934.

References

1. Abel, A.: Implementing a normalizer using sized heterogeneous types. Journal of Functional Programming 19(3-4), 287–310 (2009)
2. Abel, A.: Normalization by Evaluation: Dependent Types and Impredicativity. Habilitation Thesis (May 2013)
3. Abel, A., Chapman, J.: Normalization by evaluation in the delay monad: A case study for coinduction via copatterns and sized types. In: Levy, P., Krishnaswami, N. (eds.) Proceedings 5th Workshop on Mathematically Structured Functional Programming, Grenoble, France, 12 April 2014. Electronic Proceedings in Theoretical Computer Science, vol. 153, pp. 51–67. Open Publishing Association (2014)
4. Berger, U., Schwichtenberg, H.: An inverse of the evaluation functional for typed λ-calculus. In: Logic in Computer Science, 1991. LICS'91., Proceedings of Sixth Annual IEEE Symposium on. pp. 203–211. IEEE (1991)
5. Brady, E.C.: Idris — systems programming meets full dependent types. In: Proceedings of the 5th ACM workshop on Programming languages meets program verification. pp. 43–54. ACM (2011)
6. Coquand, C.: A formalised proof of the soundness and completeness of a simply typed lambda-calculus with explicit substitutions. Higher-Order and Symbolic Computation 15(1), 57–90 (2002)
7. Danvy, O.: Type-directed partial evaluation. Springer (1999)
8. De Bruijn, N.G.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. In: Indagationes Mathematicae (Proceedings). vol. 75, pp. 381–392. Elsevier (1972)
9. Eades, H., Stump, A.: Exploring the reach of hereditary substitution `http://metatheorem.org/wp-content/papers/cr_TYPES11_pro_submission.pdf`
10. Eades, H., Stump, A.: Hereditary substitution for stratified system f. In: International Workshop on Proof-Search in Type Theories, PSTT. vol. 10 (2010)
11. Gödel, V.K.: Über eine bisher noch nicht benützte erweiterung des finiten standpunktes. dialectica 12(3-4), 280–287 (1958)
12. Ilik, D.: Normalization of gödel's system t extended with control delimited at the type of natural numbers. ppdp tutorial. (2014), `http://www.lix.polytechnique.fr/~danko/PPDP-2014-tutorial/`
13. Joachimski, F., Matthes, R.: Short proofs of normalization for the simply-typed λ-calculus, permutative conversions and gödel's t. Archive for Mathematical Logic 42(1), 59–87 (2003)

14. Keller, C., Altenkirch, T.: Hereditary substitutions for simple types, formalized. In: Proceedings of the third ACM SIGPLAN workshop on Mathematically structured functional programming. pp. 3–10. ACM (2010)
15. Kripke, S.A.: Semantical analysis of intuitionistic logic i. Studies in logic and the foundations of mathematics 40, 92–130 (1965)
16. Magnusson, L.: The implementation of alf-a proof editor based on martin-löf's monomorphic type theory with explicit substitution (1995)
17. Martin-Löf, P.: Intuitionistic type theory. Notes by Giovanni Sambin (1984)
18. Martin-Löf, P.: An intuitionistic theory of types: Predicative part. Studies in Logic and the Foundations of Mathematics 80, 73–118 (1975)
19. Mitchell, J.C., Moggi, E.: Kripke-style models for typed lambda calculus. Annals of Pure and Applied Logic 51(1), 99–124 (1991)
20. Morrison, D.: Normalization by evaluation in agda. source code. (2009), `https://github.com/darinmorrison/nbe`
21. Nordström, B., Petersson, K., Smith, J.M.: Programming in Martin-Löf's type theory, vol. 85. Oxford University Press (1990)
22. Norell, U.: Towards a practical programming language based on dependent type theory. Chalmers University of Technology (2007)
23. Plotkin, G.D.: Lambda-definability and logical relations. School of Artificial Intelligence, University of Edinburgh (1973)
24. Prawitz, D., Deduction, N.: A proof theoretical study'. Almqvist˜ Wiksell, Stockholm (1965)
25. The Coq Development Team: The Coq Proof Assistant Reference Manual (2008), `http://coq.inria.fr`
26. Watkins, K., Cervesato, I., Pfenning, F., Walker, D.: A concurrent logical framework: The propositional fragment. In: Types for Proofs and Programs, pp. 355–377. Springer (2004)

Appendix

A Evaluation

Evaluation of expressions in NbE is like canonical evaluation of values in SbE. The proof of evaluation below does not describe how each case works, because it is essentially the same as canonical evaluation (Theorem 3 and Lemma 1 in Section 3). We invite the reader to compare the proofs of evaluation and canonical evaluation to verify their similarity. Whereas canonical evaluation is defined separately for neutrals, evaluation is a single definition for all syntactic forms.

Theorem 5 (Evaluation).

$$\mathrm{eval}^{\mathrm{E}} : \Gamma \vdash^{\mathrm{E}} A \to [\![\ \Delta \vdash \Gamma\ ]\!] \to [\![\ \Delta \vdash A\ ]\!]$$

Appendix C	Lemma 4	$\mathrm{mono}^{\mathrm{S}}$	$\forall \Delta.\ [\![\ \Xi \vdash \Gamma\ ]\!] \to [\![\ \Xi, \Delta \vdash \Gamma\ ]\!]$
Appendix A	Lemma 2	$[\![\mathrm{rec}]\!]$	$[\![\ \Gamma \vdash C\ ]\!] \to [\![\ \Gamma \vdash C \supset C\ ]\!] \to [\![\ \Gamma \vdash \mathbb{N}\ ]\!] \to [\![\ \Gamma \vdash C\ ]\!]$

Evaluation ($\mathrm{eval}^{\mathrm{E}}$) is a type-preserving translation. It takes a syntactic expression in some initial context ($\Gamma \vdash^{\mathrm{E}} A$) and a semantic environment in some terminal context ($[\![\ \Delta \vdash \Gamma\ ]\!]$), and produces a semantic value in the terminal context ($[\![\ \Delta \vdash A\ ]\!]$). The semantic environment has a semantic value for every type in the initial context, each of which must be scoped in the terminal context.

Proof. *By induction on the expression* $\Gamma \vdash^{E} A$.

Case (Zero).

$n : \Delta \vdash^{\mathrm{E}} \mathbb{N}$	(zero)
$n : [\![\ \Delta \vdash \mathbb{N}\ ]\!]$	(*definition*)

Case (Successor).

$n : \Gamma \vdash^{\mathrm{E}} \mathbb{N}$	(*assumption*)
$\sigma : [\![\ \Delta \vdash \Gamma\ ]\!]$	(*assumption*)
$n' : [\![\ \Delta \vdash \mathbb{N}\ ]\!]$	(*i.h.* $\mathrm{eval}^{\mathrm{E}}\ n\ \sigma$)
$n' : \Delta \vdash^{\mathrm{E}} \mathbb{N}$	(*definition*)
$m : \Delta \vdash^{\mathrm{E}} \mathbb{N}$	(suc n')
$m : [\![\ \Delta \vdash \mathbb{N}\ ]\!]$	(*definition*)

Case (Function).

$$
\begin{array}{ll}
b : \Gamma, A \vdash^{\mathrm{E}} B & (\textit{assumption}) \\
\sigma : [\![\ \Delta \vdash \Gamma\ ]\!] & (\textit{assumption}) \\
\Xi & (\textit{assumption}) \\
a : [\![\Delta, \Xi \vdash A]\!] & (\textit{assumption}) \\
\sigma' : [\![\ \Delta, \Xi \vdash \Gamma\ ]\!] & (\mathrm{mono}^{\mathrm{S}}\ \Xi\ \sigma) \\
b' : [\![\ \Delta \vdash B\ ]\!] & (\textit{i.h.}\ \mathrm{eval}^{\mathrm{E}}\ b\ (\sigma', a))
\end{array}
$$

Case (Variable).

$$
\begin{array}{ll}
i : \Gamma \vdash^{\mathrm{R}} A & (\textit{assumption}) \\
\sigma : [\![\ \Delta \vdash \Gamma\ ]\!] & (\textit{assumption}) \\
a : [\![\ \Delta \vdash A\ ]\!] & (\textit{lookup}\ \sigma[i])
\end{array}
$$

Case (Application).

$$
\begin{array}{ll}
f : \Gamma \vdash^{\mathrm{E}} A \supset B & (\textit{assumption}) \\
a : \Gamma \vdash^{\mathrm{E}} A & (\textit{assumption}) \\
\sigma : [\![\ \Delta \vdash \Gamma\ ]\!] & (\textit{assumption}) \\
f' : [\![\ \Delta \vdash A \supset B\ ]\!] & (\textit{i.h.}\ \mathrm{eval}^{\mathrm{E}}\ f\ \sigma) \\
f' : \forall \Delta.\ [\![\ \Gamma, \Delta \vdash A\ ]\!] \to [\![\ \Gamma, \Delta \vdash B\ ]\!] & (\textit{definition}) \\
a' : [\![\ \Delta \vdash A\ ]\!] & (\textit{i.h.}\ \mathrm{eval}^{\mathrm{E}}\ a\ \sigma) \\
b : [\![\ \Delta \vdash B\ ]\!] & (f'\ \emptyset\ a')
\end{array}
$$

Case (Primitive recursion).

$$
\begin{array}{ll}
c_z : \Gamma \vdash^{\mathrm{E}} C & (\textit{assumption}) \\
c_s : \Gamma \vdash^{\mathrm{E}} C \supset C & (\textit{assumption}) \\
n : \Gamma \vdash^{\mathrm{E}} \mathbb{N} & (\textit{assumption}) \\
\sigma : [\![\ \Delta \vdash \Gamma\ ]\!] & (\textit{assumption}) \\
c'_z : [\![\ \Delta \vdash C\ ]\!] & (\textit{i.h.}\ \mathrm{eval}^{\mathrm{E}}\ c_z\ \sigma) \\
c'_s : [\![\ \Delta \vdash C \supset C\ ]\!] & (\textit{i.h.}\ \mathrm{eval}^{\mathrm{E}}\ c_s\ \sigma) \\
n' : [\![\ \Delta \vdash \mathbb{N}\ ]\!] & (\textit{i.h.}\ \mathrm{eval}^{\mathrm{E}}\ n\ \sigma) \\
c : [\![\ \Delta \vdash C\ ]\!] & ([\![\mathrm{rec}]\!]\ c'_z\ c'_s\ n')
\end{array}
$$

□

Lemma 2 (Semantic Primitive Recursion).

$$[\![\mathrm{rec}]\!] : [\![\ \Gamma \vdash C\ ]\!] \to [\![\ \Gamma \vdash C \supset C\ ]\!] \to [\![\ \Gamma \vdash \mathbb{N}\ ]\!] \to [\![\ \Gamma \vdash C\ ]\!]$$

Appendix B	Theorem 6	reify	$[\![\ \Gamma \vdash A\ ]\!] \to \Gamma \vdash^{\mathrm{V}} A$
Appendix B	Lemma 3	$\mathrm{reflect}^{\mathrm{N}}$	$\Gamma \vdash^{\mathrm{N}} A \to [\![\ \Gamma \vdash A\ ]\!]$

Semantic primitive recursion ($[\![\mathrm{rec}]\!]$) is the lifting of the syntax of primitive recursion (rec) to the model (the semantic domain). It takes a semantic value for zero branch, a semantic value for the successor case, and a semantic value for the natural number to recurse over.

Because the model of natural numbers is just a natural number value, the model of primitive recursion works basically the same way that normalization of primitive recursion would work in the theory. The main difference is that the arguments and return type are really elements of the model. In particular, the successor branch for the primitive recursion is a model function that we may simply apply to other model values (i.e. the induction hypothesis).

Proof. *By induction on the semantic natural number value* $[\![\ \Gamma \vdash \mathbb{N}\ ]\!]$. *Definitionally, this reduces to induction on the syntactic natural number value* $\Gamma \vdash^{V} \mathbb{N}$.

Case (Zero). The zero case is immediate.

$$c_z : [\![\ \Gamma \vdash C\ ]\!] \qquad (\textit{assumption})$$

Case (Successor). In the successor case we first get the inductive hypothesis for the predecessor. Because the successor branch argument is a semantic value of function type, it is a metalanguage function. Thus, we compute the result by applying the successor branch to the empty context (weakening by nothing), and the inductive hypothesis.

$$\begin{array}{ll}
c_z : [\![\ \Gamma \vdash C\ ]\!] & (\textit{assumption}) \\
c_s : [\![\ \Gamma \vdash C \supset C\ ]\!] & (\textit{assumption}) \\
n : \Gamma \vdash^{\mathrm{V}} \mathbb{N} & (\textit{assumption}) \\
n : [\![\ \Gamma \vdash \mathbb{N}\ ]\!] & (\textit{definition}) \\
c : [\![\ \Gamma \vdash C\ ]\!] & (\textit{i.h.}\ [\![\mathrm{rec}]\!]\ c_z\ c_s\ n) \\
c_s : \forall \Delta.\ [\![\ \Gamma, \Delta \vdash C\ ]\!] \to [\![\ \Gamma, \Delta \vdash C\ ]\!] & (\textit{definition}) \\
c' : [\![\ \Gamma \vdash C\ ]\!] & (c_s\ \emptyset\ c)
\end{array}$$

Case (Neutral). If the value that we are trying to recurse over is neutral, then we reflect a syntactic primitive recursion stuck on the neutral input, while

reifying both branch arguments.

$$
\begin{array}{ll}
c_z : [\![\ \Gamma \vdash C\ ]\!] & (\textit{assumption}) \\
c_s : [\![\ \Gamma \vdash C \supset C\ ]\!] & (\textit{assumption}) \\
n : \Gamma \vdash^{\mathrm{N}} \mathbb{N} & (\textit{assumption}) \\
c'_z : \Gamma \vdash^{\mathrm{V}} C & (\text{reify } c_z) \\
c'_s : \Gamma \vdash^{\mathrm{V}} C \supset C & (\text{reify } c_s) \\
c' : \Gamma \vdash^{\mathrm{N}} C & (\text{rec } c'_z\ c'_s\ n) \\
c'' : [\![\ \Gamma \vdash C\ ]\!] & (\text{reflect } c')
\end{array}
$$

□

B Reification

Reification (Theorem 6) is the process of turning a semantic value into a syntactic value. It is defined mutually with the reflection (Lemma 3) of neutrals to semantic values.

Theorem 6 (Reification).

$$\text{reify} : [\![\ \Gamma \vdash A\ ]\!] \to \Gamma \vdash^{\mathrm{V}} A$$

Appendix B	Lemma 3	$\text{reflect}^{\mathrm{N}}$	$\Gamma \vdash^{\mathrm{N}} A \to [\![\ \Gamma \vdash A\ ]\!]$

Reification maps any semantic value ($[\![\ \Gamma \vdash A\ ]\!]$) to a syntactic value ($\Gamma \vdash^{\mathrm{V}} A$). It is mutually defined with the reflection of syntactic neutrals to semantic values ($\text{reflect}^{\mathrm{N}}$), making it possible to reify functions.

Proof. By induction on the type A.

Case (Natural numbers). Semantic natural numbers values are also syntactic values, so their reification is immediate.

$$
\begin{array}{ll}
n : [\![\ \Gamma \vdash \mathbb{N}\ ]\!] & (\textit{assumption}) \\
n : \Gamma \vdash^{\mathrm{V}} \mathbb{N} & (\textit{definition})
\end{array}
$$

Case (Functions). When reifying a function (of type $A \supset B$) we get a metalanguage function as a parameter. The function parameters are a context to weaken by and a weakened semantic value (of type A).

We would like to call the function, but we cannot a produce a semantic value of type $[\![\ \Gamma \vdash A\ ]\!]$ out of thin air. What if we make the weakening parameter of the metalanguage function be the singleton context $\emptyset, A$? Now we can call the function by making the (now weakened) parameter $[\![\ \Gamma, A \vdash A\ ]\!]$ be a variable! However, the variable must be a semantic value. Luckily, the mutually defined

lemma reflect$^{\mathrm{N}}$ can translate any syntactic neutral term (including variables) to a semantic value.

Finally, we reify the result of the metalanguage function and wrap it in a λ. Note that in this appeal to the inductive hypothesis the type gets smaller (B is smaller $A \supset B$). When reflecting the variable after weakening, the type also gets smaller (B is smaller $A \supset B$).

$f : [\![\ \Gamma \vdash A \supset B\ ]\!]$	(*assumption*)
$f : \forall \Delta.\ [\![\ \Gamma, \Delta \vdash A\ ]\!] \to [\![\ \Gamma, \Delta \vdash B\ ]\!]$	(*definition*)
$a : [\![\ \Gamma, A \vdash A\ ]\!]$	(*i.h.* reflect$^{\mathrm{N}}$ (var here))
$b : [\![\ \Gamma, A \vdash B\ ]\!]$	$(f\ (\emptyset, A)\ a)$
$b' : \Gamma, A \vdash^{\mathrm{V}} B$	(*i.h.* reify b)
$f' : \Gamma \vdash^{\mathrm{V}} A \supset B$	$(\lambda b')$

□

Lemma 3 (Neutral Reflection).

$$\mathrm{reflect}^{\mathrm{N}} : \Gamma \vdash^{\mathrm{N}} A \to [\![\ \Gamma \vdash A\ ]\!]$$

Appendix B	Theorem 6	reify	$[\![\ \Gamma \vdash A\ ]\!] \to \Gamma \vdash^{\mathrm{V}} A$

The reflection of neutrals maps any syntactic neutral ($\Gamma \vdash^{\mathrm{N}} A$) to a semantic value ($[\![\ \Gamma \vdash A\ ]\!]$). Note that we can only reflect syntactic neutrals, not syntactic values! If we could reflect syntactic values, then this would be like canonical evaluation of values without the semantic environment argument. However, the semantic environment allows the semantic values to be collected when canonically evaluating underneath a function body.

Proof. By induction on the type A.

Case (Natural numbers). The natural number case is immediate.

$n : \Gamma \vdash^{\mathrm{N}} \mathbb{N}$	(*assumption*)
$n' : \Gamma \vdash^{\mathrm{V}} \mathbb{N}$	(neut n)
$n' : [\![\ \Gamma \vdash \mathbb{N}\ ]\!]$	(*definition*)

Case (Functions). To reflect a neutral function, we get a syntactic neutral function f as a standard argument. Because we are producing a metalanguage function, we also get Δ and a as additional arguments. Reifying a changes it from a semantic value to a syntactic value. Now, we can appeal to the inductive hypothesis to reflect the *application* of a weakened version of f to the reified a. The reason why reflection of the function case is okay for neutrals, but not values, is because application of a syntactic neutral function to a syntactic value

argument produces a syntactic neutral result. Finally, note that the appeal to the inductive hypothesis is decreasing on the size of the type (where B is smaller than $A \supset B$), not the size of the term.

$$
\begin{array}{rl}
f : \Gamma \vdash^{\mathrm{N}} A \supset B & (\textit{assumption}) \\
\Delta & (\textit{assumption}) \\
a : [\![\ \Gamma, \Delta \vdash A\ ]\!] & (\textit{assumption}) \\
a' : \Gamma, \Delta \vdash^{\mathrm{V}} A & (\textit{i.h.}\ \text{reify}\ a) \\
f' : \Gamma, \Delta \vdash^{\mathrm{N}} A \supset B & (\textit{weakening}) \\
b : [\![\ \Gamma, \Delta \vdash B\ ]\!] & (\textit{i.h.}\ \text{reflect}^{\mathrm{N}}\ (f' \cdot a'))
\end{array}
$$

□

C Context Reflection

Context reflection (Theorem 7) gives a semantic meaning to a context by translating it to a semantic environment. Specifically, context reflection produces the identity semantic environment, mapping variables to themselves as semantic values. Context reflection makes use of environment monotonicity (Lemma 4), and environment monotonicity makes use of value monotonicity (Lemma 5). Both monotonicity lemmas are basically a version of weakening lifted from syntax to semantics, for environments and values respectively.

Theorem 7 (Context Reflection).

$$\text{reflect}^{\mathrm{C}} : \forall \Gamma.\ [\![\ \Gamma \vdash \Gamma\ ]\!]$$

Appendix B	Lemma 3	$\text{reflect}^{\mathrm{N}}$	$\Gamma \vdash^{\mathrm{N}} A \to [\![\ \Gamma \vdash A\ ]\!]$
Appendix C	Lemma 4	mono^{S}	$\forall \Delta.\ [\![\ \Xi \vdash \Gamma\ ]\!] \to [\![\ \Xi, \Delta \vdash \Gamma\ ]\!]$

Context reflection takes a new context (Γ), and produces a semantic environment ($[\![\ \Gamma \vdash \Gamma\ ]\!]$). At a high level, context reflection produces the semantic identity environment by reflecting each variable in the context.

Proof. *By induction on the context Γ.*

Case (Empty context). The empty context case is trivial.

$$
\begin{array}{ll}
u : \top & (\textit{trivial}) \\
u : [\![\ \emptyset \vdash \emptyset\ ]\!] & (\textit{definition})
\end{array}
$$

Case (Context extension). To produce a semantic identity environment for a context extension, we must extend the inductive hypothesis with the reflection of the variable representing the type (A) extending the context. However, the

scope (Γ, A) of the variable includes the type of the context extension (A). Thus, the semantic environment produced by the inductive hypothesis must first be weakened, using environment monotonicity (mono^{S}), before it is extended with the reflected variable.

Γ	(*assumption*)
A	(*assumption*)
$\sigma : [\![\ \Gamma \vdash \Gamma\ ]\!]$	(*i.h.* $\text{reflect}^{\text{C}}\ \Gamma$)
$\sigma' : [\![\ \Gamma, A \vdash \Gamma\ ]\!]$	($\text{mono}^{\text{S}}\ (\emptyset, A)\ \sigma$)
$x : [\![\ \Gamma, A \vdash A\ ]\!]$	($\text{reflect}^{\text{N}}$ (var here))
$\sigma'' : [\![\ \Gamma, A \vdash \Gamma, A\ ]\!]$	(σ', x)

□

Lemma 4 (Environment Monotonicity).

$$\text{mono}^{\text{S}} : \forall \Delta.\ [\![\ \Xi \vdash \Gamma\ ]\!] \to [\![\ \Xi, \Delta \vdash \Gamma\ ]\!]$$

Appendix C	Lemma 5	mono^{V}	$\forall \Delta.\ [\![\ \Gamma \vdash A\ ]\!] \to [\![\ \Gamma, \Delta \vdash A\ ]\!]$

Environment monotonicity takes a context (Δ), and a semantic environment scoped under some existing context (Ξ), and returns a semantic environment scoped under the weakening of the existing context (Δ, Ξ).

Because a semantic environment is basically a tuple of semantic values, environment monotonicity is defined by mapping value monotonicity (mono^{V}) across the tuple.

Proof. *By induction on the context* Γ.

Case (Empty context). The empty context case is trivial.

Δ	(*assumption*)
$u : \top$	(*assumption*)
$u : [\![\ \Xi, \Delta \vdash \emptyset\ ]\!]$	(*definition*)

Case (Context extension). To weaken the context extension of a semantic environment, extend the inductive hypothesis with the weakening of the semantic value extending the context by using mono^{V}.

Δ	(*assumption*)
$\sigma : [\![\ \Xi \vdash \Gamma\ ]\!]$	(*assumption*)
$x : [\![\ \Xi \vdash A\ ]\!]$	(*assumption*)
$\sigma' : [\![\ \Xi, \Delta \vdash \Gamma\ ]\!]$	(*i.h.* $\text{mono}^{\text{S}}\ \Delta\ \sigma$)
$x' : [\![\ \Xi, \Delta \vdash A\ ]\!]$	($\text{mono}^{\text{V}}\ \Delta\ x$)
$\sigma'' : [\![\ \Xi, \Delta \vdash \Gamma, A\ ]\!]$	(σ', x')

□

Lemma 5 (Value Monotonicity).

$$\text{mono}^{\text{V}} : \forall \Delta.\ [\![\ \Gamma \vdash A\ ]\!] \to [\![\ \Gamma, \Delta \vdash A\ ]\!]$$

Value monotonicity takes a new context (Δ), and a semantic value scoped under some existing context (Γ), and returns a semantic value scoped under the weakening of the existing context (Γ, Δ).

Proof. By case analysis of the type A.

Case (Natural numbers). Because semantic natural numbers are syntactic natural numbers, the natural number case can be defined via syntactic weakening.

Δ	(*assumption*)
$n : [\![\ \Gamma \vdash \mathbb{N}\ ]\!]$	(*assumption*)
$n : \Gamma \vdash^{\text{V}} \mathbb{N}$	(*definition*)
$n' : \Gamma, \Delta \vdash^{\text{V}} \mathbb{N}$	(*weakening*)
$n' : [\![\ \Gamma, \Delta \vdash \mathbb{N}\ ]\!]$	(*definition*)

Case (Functions). In the function case, we get the context Δ and the semantic function f as standard arguments. Additionally, we get a context Ξ and an argument a as arguments because we are returning a semantic function. We compute the required result by applying the semantic function f to the weakening (Δ, Ξ) and the semantic argument a. However, a and the result must be suitably altered using syntactic associativity of context concatenation.

Δ	(*assumption*)
$f : [\![\ \Gamma \vdash A \supset B\ ]\!]$	(*assumption*)
Ξ	(*assumption*)
$a : [\![(\Gamma, \Delta), \Xi \vdash A]\!]$	(*assumption*)
$a' : [\![\Gamma, (\Delta, \Xi) \vdash A]\!]$	(*associativity*)
$f : \forall \Phi.\ [\![\ \Gamma, \Phi \vdash A\ ]\!] \to [\![\ \Gamma, \Phi \vdash B\ ]\!]$	(*definition*)
$b : [\![\Gamma, (\Delta, \Xi) \vdash B]\!]$	$(f\ (\Delta, \Xi)\ a')$
$b' : [\![(\Gamma, \Delta), \Xi \vdash B]\!]$	(*associativity*)

□

D Normalization by Hereditary Substitution

Normalization by hereditary substitution (Theorem 8) defines normalization in terms of hereditary substitution in the function application and primitive recursion cases. Application by hereditary substitution (Lemma 6) is used in the

application case of normalization, and primitive recursion by hereditary substitution (Lemma 7) is used in the primitive recursion case of normalization. Primitive recursion by hereditary substitution also uses application by hereditary substitution in its successor case.

Theorem 8 (Normalization by Hereditary Substitution).

$$\mathrm{nbs} : \Gamma \vdash^{\mathrm{E}} A \to \Gamma \vdash^{\mathrm{V}} A$$

Appendix D	Lemma 6	abs	$\Gamma \vdash^{\mathrm{V}} A \supset B \to \Gamma \vdash^{\mathrm{V}} A \to \Gamma \vdash^{\mathrm{V}} B$
Appendix D	Lemma 7	rbs	$\Gamma \vdash^{\mathrm{V}} C \to \Gamma \vdash^{\mathrm{V}} C \supset C \to \Gamma \vdash^{\mathrm{V}} \mathbb{N} \to \Gamma \vdash^{\mathrm{V}} C$

Normalization by hereditary substitution takes a syntactic expression of some context (Γ) and some type (A), and returns a syntactic value of the same context and type.

Proof. *By induction on the expression $\Gamma \vdash^{E} A$.*

$$\begin{array}{lcl}
\mathrm{nbs\ zero} & \mapsto & \mathrm{zero} \\
\mathrm{nbs\ (suc}\ n) & \mapsto & \mathrm{suc\ (nbs}\ n) \\
\mathrm{nbs}\ (\lambda b) & \mapsto & \lambda(\mathrm{nbs}\ b) \\
\mathrm{nbs\ (var}\ i) & \mapsto & \mathrm{neut\ (var}\ i) \\
\mathrm{nbs}\ (f\ \cdot\ a) & \mapsto & \mathrm{abs\ (nbs}\ f)\ (\mathrm{nbs}\ a) \\
\mathrm{nbs\ (rec}\ c_z\ c_s\ n) & \mapsto & \mathrm{rbs\ (nbs}\ c_z)\ (\mathrm{nbs}\ c_s)\ (\mathrm{nbs}\ n)
\end{array}$$

All cases except for the eliminations translate expressions to values immediately or via congruences. The application and primitive recursion cases are handled by lemmas that work on values (abs and rbs respectively).

□

Lemma 6 (Application by Hereditary Substitution).

$$\mathrm{abs} : \Gamma \vdash^{\mathrm{V}} A \supset B \to \Gamma \vdash^{\mathrm{V}} A \to \Gamma \vdash^{\mathrm{V}} B$$

Section 2	Theorem 2	sbe	$\Gamma \vdash^{\mathrm{V}} A \to \Delta \vdash^{\mathrm{V}} \Gamma \to \Delta \vdash^{\mathrm{V}} A$

Application by Hereditary Substitution takes a syntactic function value (of type $A \supset B$) and a syntactic value for the domain of the function (of type A), and returns a syntactic value for the codomain of the function (of type B).

Proof. *By induction on the function value $\Gamma \vdash^{V} A \supset B$.*

$$\begin{array}{lcl}
\mathrm{abs}\ (\lambda b)\ a & \mapsto & \mathrm{sbe}\ b\ (\sigma_{\mathrm{id}}, a) \\
\mathrm{abs\ (neut}\ f)\ a & \mapsto & \mathrm{neut}\ (f\ \cdot\ a)
\end{array}$$

The neutral case is a congruence, and the function case is defined by hereditary substitution of the argument into the function body. Specifically, the sbe is called by extending the identity syntactic environment σ_{id} with the argument a.

□

Lemma 7 (Primitive Recursion by Hereditary Substitution).

$$\text{rbs} : \Gamma \vdash^{\text{V}} C \to \Gamma \vdash^{\text{V}} C \supset C \to \Gamma \vdash^{\text{V}} \mathbb{N} \to \Gamma \vdash^{\text{V}} C$$

Appendix D	Lemma 6	abs	$\Gamma \vdash^{\text{V}} A \supset B \to \Gamma \vdash^{\text{V}} A \to \Gamma \vdash^{\text{V}} B$

Primitive Recursion by Hereditary Substitution recurses over a syntactic natural number value ($\Gamma \vdash^{\text{V}} \mathbb{N}$) with two syntactic branch values ($\Gamma \vdash^{\text{V}} C$ and $\Gamma \vdash^{\text{V}} C \supset C$), and returns a syntactic value ($\Gamma \vdash^{\text{V}} C$).

Proof. *By induction on the natural number value* $\Gamma \vdash^{V} \mathbb{N}$.

$$\begin{array}{lcl} \text{rbs } c_z\ c_s\ \text{zero} & \mapsto & c_z \\ \text{rbs } c_z\ c_s\ (\text{suc } n) & \mapsto & \text{abs } c_s\ (\text{rbs } c_z\ c_s\ n) \\ \text{rbs } c_z\ c_s\ (\text{neut } n) & \mapsto & \text{neut } (\text{rec } c_z\ c_s\ n) \end{array}$$

The zero case is immediate and the neutral case is a congruence. The successor case applies the successor branch to the inductive hypothesis using abs (Lemma 6).

□