JAIST Repository
https://dspace.jaist.ac.jp/

Title	Normalization by stack-based evaluation
Author(s)	Vestergaard, Rene
Citation	Research report (School of Information Science, Japan Advanced Institute of Science and Technology), IS-RR-2006-010: 1-36
Issue Date	2006-07-07
Type	Technical Report
Text version	publisher
URL	http://hdl.handle.net/10119/8409
Rights	
Description	リサーチレポート（北陸先端科学技術大学院大学情報科学研究科）

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

Normalisation by Stack-Based Evaluation

René Vestergaard
JAIST, Nomi, Japan
<vester@jaist.ac.jp>

IS-RR-2006-010

July 7, 2006

Normalisation by Stack-Based Evaluation

René Vestergaard
JAIST, Nomi, Japan
<vester@jaist.ac.jp>

Abstract

We introduce Normalisation by Stack-based Evaluation, NbSE, combining Normalisation by Evaluation, NbE, with stack-based evaluation *à la* ZINC and Krivine's Machine. Stack-based evaluation avoids building, returning, and unpacking some of the *closures* (i.e., code, environment pairs) that evaluation uses internally as primitives. NbE makes evaluation two-sorted and invokes sort-coercers, called *reify* and *reflect*, in order to eliminate (external) closures from the domain of yielded values. We show that their combination, NbSE, can be implemented entirely without closures for both the call-by-name and call-by-value paradigms. The call-by-value implementation is more aggressive than usual in that it fully normalises argument terms before passing them on. The technical development in the paper establishes a conservative-extension hierarchy of the considered evaluation mechanisms that focuses on type soundness and its correctness consequences; this includes the various NbE/NbSE theorems.

1 Introduction

When implementing a programming language, the engendered execution speed of programs is a typical concern. In the case of functional (i.e., λ-calculus derived) languages, see Figure 1, a main obstacle to speed is their higher-order nature that allows for *reduction* to take place anywhere in a term and for new reduction needs, i.e., *redexes*, to be created anywhere. Firstly, it is expensive to repeatedly search for redexes. Secondly, it is expensive to perform the *substitution* that is incurred by *contracting* a redex. Instead of implementing full reduction relations, it is therefore standard to merely implement the *evaluation* part of a relation.

1.1 Evaluation

To evaluate means to only consider redexes at the outermost level of a term, so to speak. Evaluation has the dual benefits (i) of eliminating the need for redex-searching because only pre-determined positions are considered contractible and (ii) of postponing substitutions — through the use of threaded *environments*,

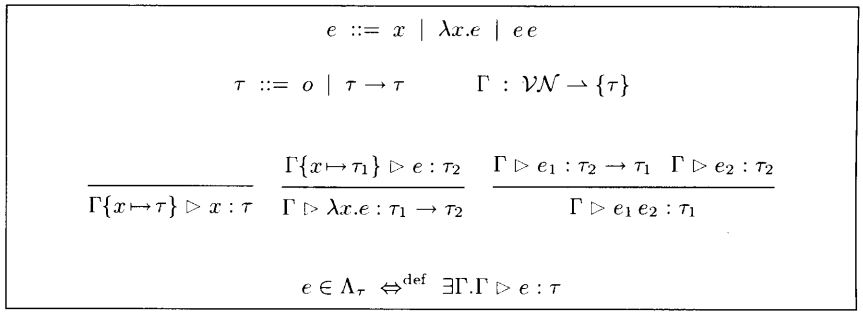


Figure 1: The implicitly simply typed λ-calculus — o is a ground type, $\Gamma\{x \mapsto \tau\}$ is the function Γ extended with/shadowed by the mapping of x to τ.

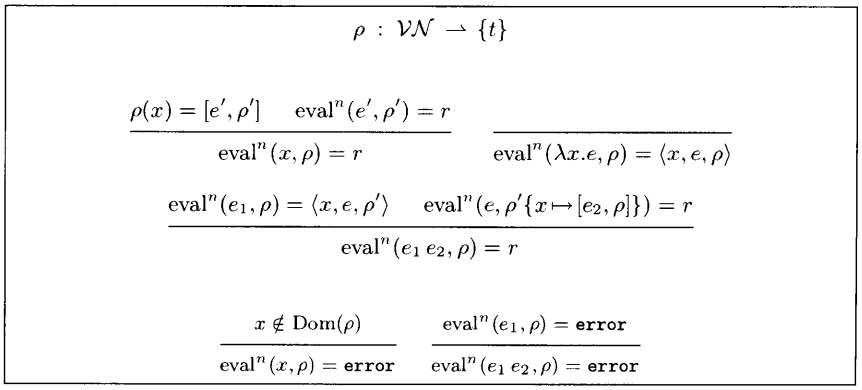


Figure 2: Big-step semantics for CbN-evaluation

ρ — until the target itself is being evaluated. The trade-off is that terms yield *run-time values*, r, rather than terms, when evaluated.

$$\begin{array}{rcl} k & ::= & \langle x, e, \rho \rangle \\ t & ::= & [e, \rho] \\ r & ::= & k \mid \texttt{error} \end{array}$$

We use k and t to range over *closures* and *thunks*, respectively. Closures record an evaluation state that has been aborted because an abstraction does not contain any "outermost" redexes. Thunks record suspended computations in a more general sense. The constant `error` is used to account for attempted computation that is not well-defined, i.e., a run-time error. That said, we show that `error` is not yielded as a result when starting from a certain class of terms (essentially closed, simply-typed terms) relative to any of the evaluation mechanisms we consider in this paper.

Refer to Figures 2 and 3 for two big-step semantics (roughly: functions de-

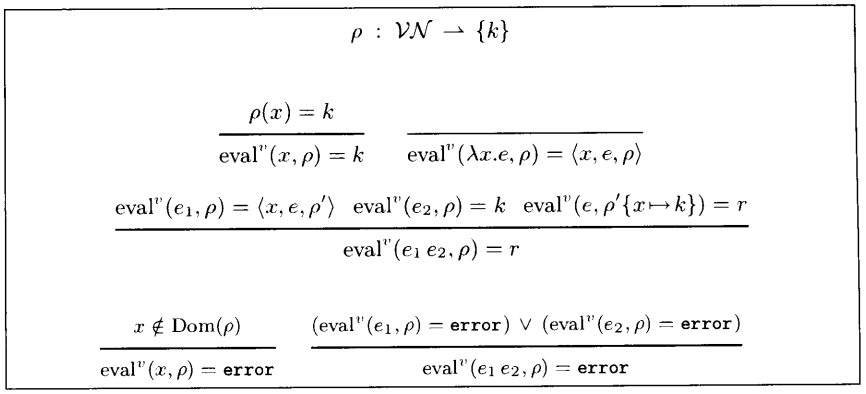


Figure 3: Big-step semantics for CbV-evaluation

fined over the abstract syntax) for call-by-name (CbN) and call-by-value (CbV) evaluation of the λ-calculus, respectively. The main difference between the two is their treatment of the argument in the application case, which results in the indicated differences in their use of environments. For CbV, arguments are evaluated before being passed to the applied function, which means that the environments, ρ, store closures, k. For CbN, arguments are passed to functions as they are and environments thus store thunks, t. With this, we can note that evaluation will yield at most one result for a given term, environment pair.

Proposition 1 (Well-definedness) $\text{eval}^s(-,-)$ *is functional, for* $s \in \{n, v\}$.

Proof The case-splitting (over e and ρ) is non-overlapping in each figure. □

We also note that all errors are explicitly accounted for in the following sense.

Proposition 2 (Quasi-totality) *If, for* $s \in \{n, v\}$, $\text{eval}^s(e, \rho)$ *is undefined, the evaluation of* e *relative to* ρ *does not terminate.*

Proof The case-splitting (over e and ρ) is exhaustive in each figure. □

The proposition, in other words, rules out that eval^s may be undefined because no rule can be applied to a particular combination of arguments.

1.2 Stack-Based Evaluation

Refining the standard notion of evaluation, we can consider states of the form (e, ρ, δ) with a term, e, to be evaluated in the context of an environment, ρ, as well as a stack, δ. The new feature, the stack, is used to maintain the application context (or *spine*) of the term that is being evaluated.

$$((e\, e_1)\, e_2, \rho, \delta) \twoheadrightarrow (e, \rho, \delta \cdot a_2 \cdot a_1)$$

The a_i's are the values denoted by the arguments, e_i, under ρ and an empty stack, and $\delta \cdot a_2 \cdot a_1$ is the stack obtained by pushing on the values in the

order they are encountered, i.e., outermost-first. When the stack is empty, the evaluation step for an abstraction remains unchanged: a closure is built and returned. When the stack is non-empty, however, evaluation can directly bind the abstracted variable to the top element of the stack, i.e., the last encountered argument.

$$(\lambda x.e, \rho, \delta \cdot a) \rightarrow (e, \rho\{x \mapsto a\}, \delta)$$

In the non-empty stack case, several administrative steps are thus avoided. Consider, for example, the following stack-based evaluation sequence.

$$\begin{aligned}
(((\lambda x_1.\lambda x_2.e_0)\, e_1)\, e_2, \rho, \delta) &\rightarrow ((\lambda x_1.\lambda x_2.e_0)\, e_1, \rho, \delta \cdot a_2) \\
&\rightarrow (\lambda x_1.\lambda x_2.e_0, \rho, \delta \cdot a_2 \cdot a_1) \\
&\rightarrow (\lambda x_2.e_0, \rho\{x_1 \mapsto a_1\}, \delta \cdot a_2) \\
&\rightarrow (e_0, \rho\{x_1 \mapsto a_1\}\{x_2 \mapsto a_2\}, \delta)
\end{aligned}$$

Stack-free evaluation, on the other hand, will build, return, and unpack (in close succession) the following two closures: $\langle x_1, \lambda x_2.e_0, \rho\rangle$, $\langle x_2, e_0, \rho\{x_1 \mapsto a_1\}\rangle$, in addition to performing the work described above. The expectation is, therefore, that stack-based evaluation will have lower administrative overhead than stack-free evaluation, i.e., will result in increased evaluation speed. We will define stack-based evaluation formally in Section 3.

1.3 Normalisation by Evaluation

Normalisation by Evaluation (NbE) has, as the name states, the effect of fully normalising terms. This should be contrasted with standard evaluation technology that yields closures as results, rather than (normal-formed) terms. NbE employs standard evaluation machinery but, in addition, relies on a canonical mechanism for supplying function-typed objects with a generic argument that allows evaluation to continue when it would otherwise stop. NbE works by distinguishing between "semantical" objects, which include the terms we ultimately are interested in normalising, and "syntactical" normal forms, which will be the computation results.[1] For variables, we let x and y divide $\mathcal{VN}$

$$\begin{aligned}
\overline{\mathcal{VN}}_x \cap \underline{\mathcal{VN}}_y &= \emptyset \\
\overline{\mathcal{VN}}_x \cup \underline{\mathcal{VN}}_y &= \mathcal{VN} \\
|\, \overline{\mathcal{VN}}_x \,| &= |\, \underline{\mathcal{VN}}_y \,|
\end{aligned}$$

And, for terms, we consider two levels in the following sense.

$$\begin{aligned}
e &::= x \mid \overline{\lambda} x.e \mid e\, \overline{@}\, e \\
&\mid d \\
c &::= \underline{\lambda} y.c \mid \underline{\text{body}}\ d \\
&\mid e \\
d &::= y \mid d\, \underline{@}\, c
\end{aligned}$$

[1] The terminology refers to a particular NbE application, namely *de-compilation*.

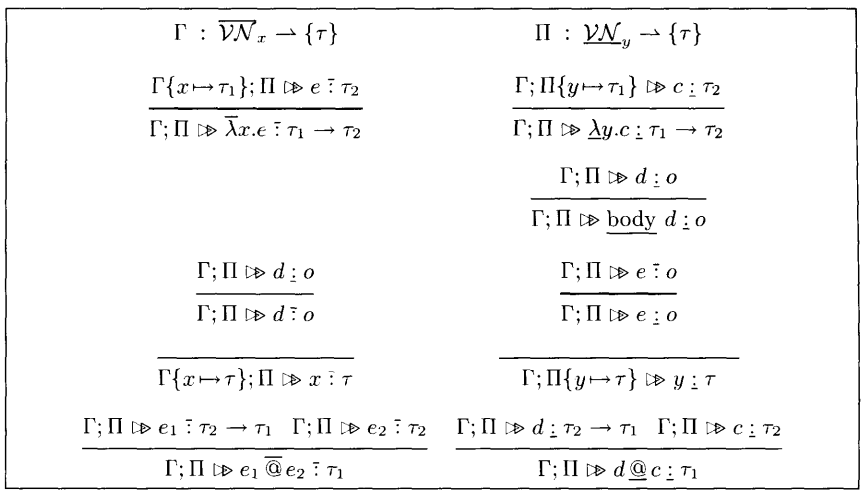


Figure 4: Simple, two-level types for an NbE λ-calculus — the left column types the "semantics", the right the "syntax"

The terms consist of a full, overlined copy of the λ-terms: e, the "semantics" as it were, and an underlined copy of "syntax": c, which amounts to long $\beta(\eta)$-normal forms, $\Lambda_\tau^{\text{long }\beta(\eta)}$, at type τ [1]. For convenience, we shall sometimes ignore the distinction between the syntactic sorts ranged over by c and d (i.e., syntactic constructors and deconstructors, respectively).

$$b \quad ::= \quad c \mid d$$

As shown in [2], these terms lend themselves to a natural presentation of NbE when constrained by a corresponding two-level type system enforcing that the overlaps between the term-sorts ranged over by e, c, and d are at ground type, see Figure 4.

$$\begin{aligned}
e \in \overline{\Lambda}_\tau^{\text{NbE}} &\Leftrightarrow^{\text{def}} \exists \Gamma, \Pi \,.\, \Gamma; \Pi \rhd\!\!> e \,\overline{:}\, \tau \\
c \in \underline{C}_\tau &\Leftrightarrow^{\text{def}} \exists \Gamma, \Pi \,.\, \Gamma; \Pi \rhd\!\!> c \,\underline{:}\, \tau \\
d \in \underline{D}_\tau &\Leftrightarrow^{\text{def}} \exists \Gamma, \Pi \,.\, \Gamma; \Pi \rhd\!\!> d \,\underline{:}\, \tau \\
b \in \underline{B}_\tau &\Leftrightarrow^{\text{def}} b \in \underline{C}_\tau \cup \underline{D}_\tau
\end{aligned}$$

The ground-type restriction on the *body*-construct guarantees long $\beta(\eta)$ normal formedness of the underlined level. The ground-type restrictions on d in e and e in c are what ultimately guarantee that we perform *normalisation* as they prevent (higher-typed) closures — that can contain outstanding inner computation — from being first-class objects in a general sense. The fact that we perform normalisation amounts to the property that an underlined term contains no overlined syntax, i.e., that a term is equal to the underlined embedding,

$\llcorner - \lrcorner$, of the standard λ-term obtained by stripping away all lining, $\| - \|$.

$$b \in \underline{B}_\tau^{=} \quad \Leftrightarrow^{\text{def}} \quad b \in \underline{B}_\tau \ \wedge \ b = \llcorner \| \, b \, \| \lrcorner$$

Proposition 3 $\llcorner \Lambda_\tau^{\text{long } \beta(\eta)} \lrcorner = \underline{B}_\tau^{=} \cap \underline{C}_\tau$

The technology that ensures that the terms above can be used for NbE are the following two functions, i.e., the canonical type-indexed coercers between the two considered term levels.

Proposition 4 (Reify and Reflect are Level Coercers) *Defined as follows, $\downarrow^-$ and $\uparrow_-$ are total, computable functions on τ and they send well-typed terms to well-typed terms, as prescribed.*

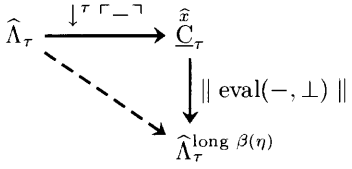
$$\begin{array}{rcll} \downarrow^\tau & : & \overline{\Lambda}_\tau^{\text{NbE}} \longrightarrow \underline{C}_\tau & \text{(aka reify)} \\ \downarrow^o e & = & e & \\ \downarrow^{\tau_1 \to \tau_2} e & = & \underline{\lambda} y. \downarrow^{\tau_2} (e \, \overline{@} \, (\uparrow_{\tau_1} y)) & (\text{for } y \notin \text{FV}(e)) \end{array}$$

$$\begin{array}{rcll} \uparrow_\tau & : & \underline{D}_\tau \longrightarrow \overline{\Lambda}_\tau^{\text{NbE}} & \text{(aka reflect)} \\ \uparrow_o d & = & d & \\ \uparrow_{\tau_1 \to \tau_2} d & = & \overline{\lambda} x. \uparrow_{\tau_2} (d \, \underline{@} \, (\downarrow^{\tau_1} x)) & (\text{for } x \notin \text{FV}(d)) \end{array}$$

Proof $\downarrow^-$ and $\uparrow_-$ are defined by structural recursion over the (well-founded) inductive data-type of τ, which means that they are total, computable functions by construction. The type-preservation property follows by a straightforward rule induction in the definition of $\downarrow^-$ and $\uparrow_-$. $\square$

Informally speaking, reify and reflect wrap their term arguments in a term context that is uniquely determined by the type argument in such a way that the result exists in the other term and typing level. The term argument itself is not changed, or even copied or discarded.

If we define $\ulcorner - \urcorner$ to completely overline an ordinary λ-term, $\| - \|$ to strip off all lining, $\widehat{\ }(\widehat{x})$ to denote $(x$-)closed terms, eval$(-,-)$ to be evaluation, and $\perp$ to be the empty environment, we can describe the workings of NbE as the commutation of the following diagram, with the dashed arrow being (inferred) normalisation.



The key to understanding the diagram lies in the fact that evaluation, when constrained by $\downarrow^\tau \ulcorner - \urcorner$, removes any and all overlined objects from $\underline{\widehat{C}}_\tau^{\widehat{x}}$, as we shall see in Section 4. Ultimately, this is so because there are no closed, normal-formed terms of ground type and because all overlined content of a $\underline{\widehat{C}}_\tau^{\widehat{x}}$-term by construction is closed and of ground type at the outermost.

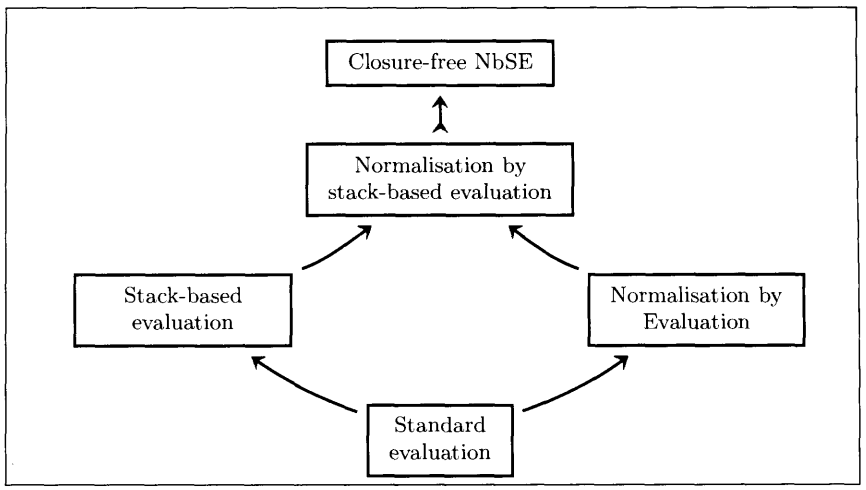


Figure 5: Normalisation by stack-based evaluation as a conservative extension of stack-based evaluation, Normalisation by Evaluation, and standard evaluation, as well as its closure-free implementation

1.4 Our Contribution

Figure 5 presents the evaluation-mechanism hierarchy we establish in this paper. We use $\rightarrow$ to express that the origin mechanism *is conservatively extended by* the target. Informally speaking, this means that everything that can done at the base of the arrow can be done at the head, albeit with more notation/formal machinery. Properties concerning the correctness of the different mechanisms can therefore be transferred between the systems. We also derive a non-trivial implementation of vanilla NbSE, as expressed by $\rightarrowtail$ in the figure, which is closure-free and, thus, potentially very efficient. In addition to the conceptual contribution of the figure itself, this paper contributes technically by cleanly introducing NbSE and pointing out that it can implemented to be closure-free in a similarly clean manner.

1.5 Relevance

Our technical presentation is narrowly focused on the use of the considered formalism for real-time simplification of λ-terms, e.g., in an interpreter. The specific application we have in mind is to use the considered big-step semantics as the internal reduction engine in a type-theory based theorem prover, for which the computed results must be λ-terms (as opposed to closures) to allow the person doing the proving to read the generated proofs.

$$
\begin{array}{lll}
\models^{\triangleright} \rho : \Gamma & \Leftrightarrow^{\mathrm{def}} & \left\{ \begin{array}{l} \mathrm{Dom}(\rho) = \mathrm{Dom}(\Gamma) \wedge \\ \forall x \in \mathrm{Dom}(\rho).\ \models^{\triangleright} \rho(x) : \Gamma(x) \end{array} \right. \\
\models^{\triangleright} [e, \rho] : \tau & \Leftrightarrow^{\mathrm{def}} & \exists \Gamma.\ \models^{\triangleright} \rho : \Gamma \wedge \Gamma \triangleright e : \tau \\
\models^{\triangleright} \langle x, e, \rho \rangle : \tau & \Leftrightarrow^{\mathrm{def}} & \exists \Gamma.\ \models^{\triangleright} \rho : \Gamma \wedge \Gamma \triangleright \lambda x.e : \tau
\end{array}
$$

Figure 6: Simple value typing

2 Evaluation

This section introduces the basic formal framework we employ in the paper. It will be done in the context of the standard, stack-free evaluation mechanisms for the λ-calculus, see Figures 2 and 3. The core technical concepts we need are those of *value typing* and *soundness*, from which we are able to conclude (minimal) correctness fairly straightforwardly. We state the soundness result for (standard) evaluation although, as suggested in the introduction, we shall not prove it in a stand-alone manner. Instead, it will follow from our formal treatment of NbSE, see Appendix B.

2.1 Value Typing

We saw that evaluation should be properly thought of as computing on states rather than on terms. We therefore extend simple typing to environments and their content: thunks and closures. In other words, we define *value typing* (by a triple induction, anchored in simple typing of terms).

Definition 5 *Simple value typing, $\models^{\triangleright}$, is defined in Figure 6.*

Writing $\not\models^{\triangleright}$ for the complement of $\models^{\triangleright}$, we state an immediate consequence of the definition.

Proposition 6 $\not\models^{\triangleright} \mathtt{error} : \tau$

2.2 Value vs Simple Typing

A subtlety in the definition of value typing pertains to the use of existentially quantified typing environments, Γ, for closures and thunks. Because of this, it is possible for a variable name that occurs freely in different bindings in a given environment, ρ, e.g., $\{y \mapsto [x, -]\}\{z \mapsto [x, -]\}$ to have its occurrences typed differently, i.e., by an intersection type from a global perspective, which may suggest that the simple-typing discipline is broken. However, each occurrence will itself be bound in a term environment and, indeed, all free variables will ultimately have to be bound to a closed term to terminate the $\models^{\triangleright}$-nesting. This means that the "intersection-typed" variables are in line to be replaced by terms whose types obviously can be different while retaining simple typability of the overall term. Another way of saying this is that the variables are to be considered different (in terms of binding) and we will never be forced to write them next to

each other in a term (where they would have the same binding). To illustrate this point a bit further, we define the action of an environment on a term as follows — in the last case, $-\langle - := - \rangle$ is meta-level, capture-avoiding substitution, with all its formal short-comings [3], and z is a (not-further-specified) variable that is fresh with respect to $\lambda x.e$ and ρ.

$$\begin{array}{rcl} \rho[x] & =^{\text{def}} & \begin{cases} \rho'[e] & \text{if } \rho(x) = [e, \rho'] \\ \rho'[\lambda y.e] & \text{if } \rho(x) = \langle y, e, \rho' \rangle \\ x & \text{otherwise} \end{cases} \\ \rho[e_1\, e_2] & =^{\text{def}} & \rho[e_1]\, \rho[e_2] \\ \rho[\lambda x.e] & =^{\text{def}} & \lambda z.\rho[e\langle x := z \rangle] \end{array}$$

An environment acting on a term will repeatedly seek to apply all the substitutions that it and its bindings prescribe. The point, which we note without proof, is that value typing implies simple typing under environment action.

$$\begin{array}{rcl} \models^{\triangleright} [e, \rho] : \tau & \Rightarrow & \bot \rhd \rho[e] : \tau \\ \models^{\triangleright} \langle x, e, \rho \rangle : \tau & \Rightarrow & \bot \rhd \rho[\lambda x.e] : \tau \end{array}$$

We also note that this implies that value typing only can be applied to terms that "morally" are closed.

2.3 Soundness

The main technical result we establish for the various evaluation mechanisms is (type) soundness. It states an equivalent property to that of subject reduction for reduction relations, namely that types are preserved by computation. The result employs value typing, which, in fact, has been introduced for this exact purpose. Soundness states that a well-typed term with a correspondingly value-typed environment will result in a similarly value-typed result, provided evaluation terminates.[2]

Lemma 7 (Soundness) *Let $s \in \{n, v\}$, see Figures 2 and 3.*

$$\begin{array}{l} \Gamma \rhd e : \tau \wedge \models^{\triangleright} \rho : \Gamma \wedge \text{eval}^s(e, \rho) = r \\ \Downarrow \\ \models^{\triangleright} r : \tau \end{array}$$

Proof See Appendix B.1. □

[2]We stress that termination means that evaluation yields an arbitrary run-time value, r. Insisting on the more restrictive possibility that a closure, k, is yielded would prevent us from obtaining any useful kind of correctness property from soundness, see Theorem 8.

Apart from functioning as a basic sanity check, the lemma also plays a part in establishing correctness of evaluation, as we shall see next.

2.4 Error-Freeness

The notion of correctness we concern ourselves with is minimal in nature and addresses neither termination nor preservation of $\alpha\beta\eta$-equivalence up to environment actions (although these properties, of course, would be relevant for separate reasons and, indeed, do hold). Instead, we focus narrowly on what we can derive from soundness, namely that well-typed computation "does not go wrong". For the ensuing notions of evaluation, we shall see that this notion of correctness becomes gradually more interesting.

Theorem 8 (Error-Freeness) *Let $s \in \{n, v\}$, see Figures 2 and 3.*

$$\bot \rhd e : \tau \;\Rightarrow\; \mathrm{eval}^s(e, \bot) \neq \texttt{error}$$

Proof By Proposition 6 and Lemma 7. □

The inequality in the theorem, $\neq$, is the complement of equality, which means that the result allows for $\mathrm{eval}^s(e, \bot)$ to be undefined. With the explicit, non-value-typable `error` run-time value to compare against, the above result implies that we can characterise what correctness means in terms of the behaviour of the evaluation functions.

Proposition 9 (Digest) *Well-typed evaluation will*

- *yield a closure when applied to the function position of an application (if it terminates) and, more generally,*
- *not yield* `error` *from any sub-computation, and, finally,*
- *bind any encountered variable in the associated environment.*

Proof Theorem 8 juxtaposed with the rules that generate and propagate `error` in Figures 2 and 3. □

3 Stack-Based Evaluation

In this section, we account for stack-based evaluation in terms of the formal framework we outlined in the previous section. The definitions of the relevant call-by-name and call-by-value big-step semantics are recalled in Figures 7 and 8, respectively. It is interesting to note that, as a result of the "optimisation" of their use of closures, these two mechanisms have one less `error`-rule each compared to the corresponding mechanisms for standard evaluation, see Figures 2 and 3, respectively.

Proposition 10 (Well-definedness) $\mathrm{Seval}^s(-,-,-)$ *is functional, for $s \in \{n, v\}$.*

Proof The case-splitting is non-overlapping in each figure. □

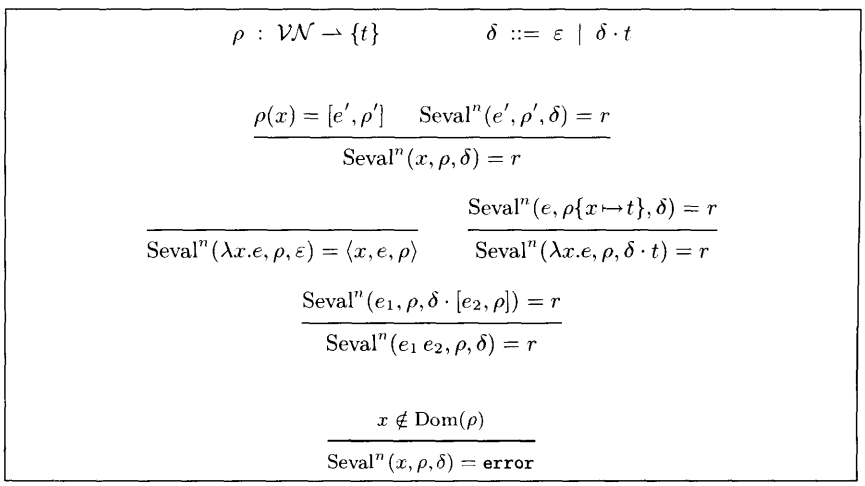


Figure 7: Big-step semantics for stack-based CbN-evaluation

Proposition 11 (Quasi-totality) *If, for* $s \in \{n, v\}$, $\text{Seval}^s(e, \rho, \delta)$ *is undefined, the evaluation of* e *relative to* ρ *and* δ *does not terminate.*

Proof The case-splitting is exhaustive in each figure. □

3.1 Type Theory

In order to address typing in the presence of a stack, we note that the considered evaluation mechanism implements a sort of uncurrying that is not explicitly captured by the traditional type system for the λ-calculus, see Figure 1. Instead, we can introduce the obvious notion of *stack type*, as a stack of types, see Figure 9: Δ, as well as notation for the type of a function that can be applied to a stack: $\Delta \Rightarrow \tau$. We write ε for the empty stack type, $\Delta \cdot \tau$ for the stack type obtained from Δ by pushing τ on the top, τ as short-hand for $\varepsilon \cdot \tau$, $\Delta\Delta'$ for the "stacking" of two stack types, and, finally, we use $\Delta.i$ to denote the ith element of the stack type, e.g., $(\Delta \cdot \tau).1 = \tau$. We use δ to denote an actual stack with essentially the same notation as for stack types. The notation $\Delta \Rightarrow \tau$ is defined inductively over stack types as follows.

$$\begin{aligned} \varepsilon \Rightarrow \tau &=^{\text{def}} \tau \\ (\Delta \cdot \tau_1) \Rightarrow \tau_2 &=^{\text{def}} \tau_1 \rightarrow (\Delta \Rightarrow \tau_2) \end{aligned}$$

A term will be stack-typed relative to a (term) environment, Γ, and a stack type, Δ, resulting in typing judgements of the following form, see Figure 9:

$$\Gamma \mid \Delta \Rrightarrow e : \tau$$

The above e is meant to produce a value of type τ when encountered in a context with arguments of type Δ on the accompanying stack. When not all

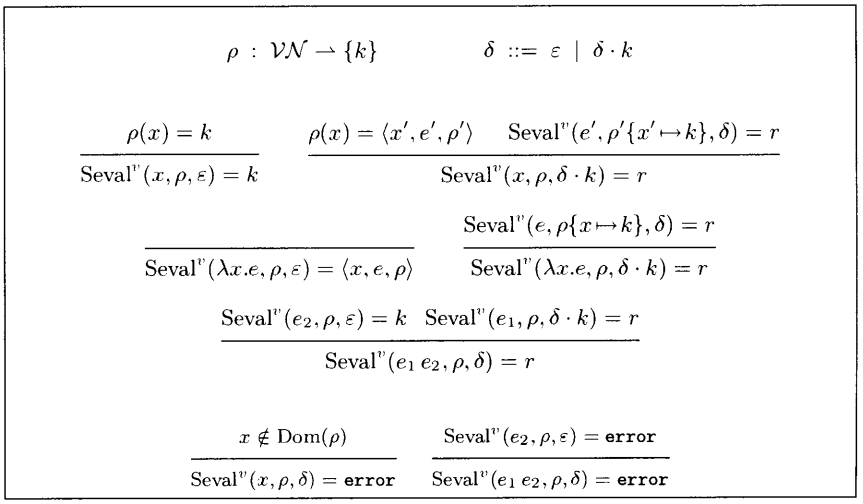


Figure 8: Big-step semantics for stack-based CbV-evaluation

arguments to a given function are supplied and the function itself is passed on as an argument, we pack the stack type into an ordinary function type with the use of the (abs)-rule in Figure 9. In turn, the (code)-rule in the figure is intended to be used to unpack an ordinary function type to a stack type when a function that has been supplied as (a higher-order) argument is about to be applied. Intuitively, stack-typing is a concise way of differentiating the dual roles of abstractions as first-class objects: as code that can be executed and as code that is treated as data. The preceding discussion, however, suggests that, from a technical perspective, the use of stack types is inconsequential. In other words, and more formally speaking, we have that stack-based and simple typing are equivalent.

Lemma 12 *See Figures 1 and 9.*

$$\Gamma \rhd e : \Delta \Rightarrow \tau \;\Leftrightarrow\; \Gamma \,|\, \Delta \rhd\!\!\rhd e : \tau$$

Proof Straightforward rule inductions. □

The result implies that stack-based typing is a conservative extension of simple typing.

Corollary 13 *See Figures 1 and 9.*

$$\Gamma \rhd e : \tau \;\Leftrightarrow\; \Gamma \,|\, \varepsilon \rhd\!\!\rhd e : \tau$$

As for value typing, we can straightforwardly adapt Definition 5 to address stack types and to pertain to stack-based typing more generally.

Definition 14 *Stack-based value typing, $\models^{\rhd\!\!\rhd}$, is defined in Figure 10.*

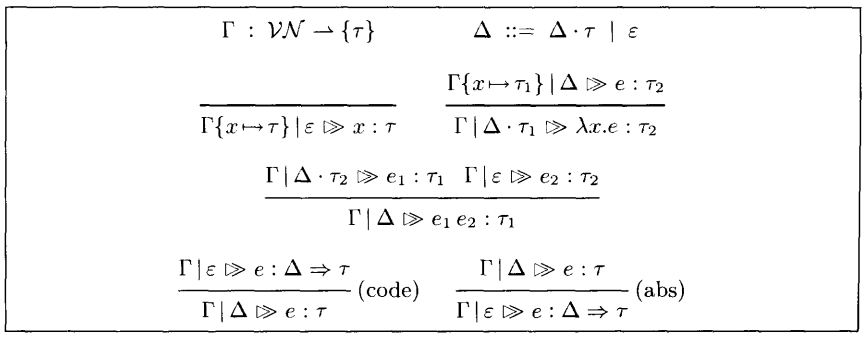


Figure 9: Simple stack-based typing

$$
\begin{array}{rcl}
\models^{\rhd} \rho : \Gamma & \Leftrightarrow^{\text{def}} & \left\{ \begin{array}{l} \mathrm{Dom}(\rho) = \mathrm{Dom}(\Gamma) \wedge \\ \forall x \in \mathrm{Dom}(\rho).\ \models^{\rhd} \rho(x) : \Gamma(x) \end{array} \right. \\
\models^{\rhd} \delta : \Delta & \Leftrightarrow^{\text{def}} & |\delta| = |\Delta| \wedge \forall i \in \{1, \ldots |\delta|\}.\ \models^{\rhd} \delta.i : \Delta.i \\
\models^{\rhd} [e, \rho] : \tau & \Leftrightarrow^{\text{def}} & \exists \Gamma.\ \models^{\rhd} \rho : \Gamma \wedge \Gamma \mid \varepsilon \rhd\!\!> e : \tau \\
\models^{\rhd} \langle x, e, \rho \rangle : \tau & \Leftrightarrow^{\text{def}} & \exists \Gamma.\ \models^{\rhd} \rho : \Gamma \wedge \Gamma \mid \varepsilon \rhd\!\!> \lambda x.e : \tau
\end{array}
$$

Figure 10: Stack-based value typing

Proposition 15 $\not\models^{\rhd} \texttt{error} : \tau$

Unsurprisingly, perhaps, we have that also value typing for the simple and the stack-based disciplines coincide.

Lemma 16

$$
\begin{array}{rcll}
(\models^{\triangleright} \rho : \Gamma & \Leftrightarrow & \models^{\rhd} \rho : \Gamma) & \wedge \\
(\models^{\triangleright} [e, \rho] : \tau & \Leftrightarrow & \models^{\rhd} [e, \rho] : \tau) & \wedge \\
(\models^{\triangleright} \langle x, e, \rho \rangle : \tau & \Leftrightarrow & \models^{\rhd} \langle x, e, \rho \rangle : \tau)
\end{array}
$$

Proof Straightforward triple rule inductions, using Corollary 13. □

3.2 Operational Semantics

While it was straightforward to show that standard and stack-based (value) typing coincide, it is a bit more subtle to establish a connection between evaluation with and without a stack. The required lemmas, as it turns out, are the following two "continuation" properties for stack-based evaluation in the cases of computation failure and success, respectively.

Lemma 17 *Let* $s \in \{n, v\}$, *cf. Figures 7 and 8.*

$$\mathrm{Seval}^s(e, \rho, \delta) = \texttt{error} \Rightarrow \mathrm{Seval}^s(e, \rho, \delta'\delta) = \texttt{error}$$

Proof By rule induction in Seval^s. □

Lemma 18 *Let $s \in \{n, v\}$ and let l^s be t in case $s = n$ and k in case $s = v$.*

$$\begin{array}{l} \mathrm{Seval}^s(e, \rho, \delta) = \langle z, e_0, \rho_0 \rangle \ \wedge \ \mathrm{Seval}^s(e_0, \rho_0\{z \mapsto l^s\}, \varepsilon) = r \\ \Downarrow \\ \mathrm{Seval}^s(e, \rho, l^s\delta) = r \end{array}$$

Proof By rule induction in Seval^s. The cases where the stack is not ε follow by trivial invocations of the I.H.. The cases with ε can be simulated by the corresponding rule with a non-empty stack, by assumption. □

With these properties in place, we can show that stack-based evaluation is, indeed, a conservative extension of standard, or stack-free, evaluation.

Lemma 19 *Let $s \in \{n, v\}$, cf. Figures 2, 3, 7, and 8.*

$$\mathrm{eval}^s(e, \rho) = r \quad \Rightarrow \quad \mathrm{Seval}^s(e, \rho, \varepsilon) = r$$

Proof By straightforward rule inductions in eval^s, using Lemmas 17 and 18 in the application cases. □

We stress that the above property involves an r, rather than just a k, which means that no errors are introduced or overcome by the use of a stack (except for any non-terminating cases). More generally speaking, the lemma establishes that the considered stack correctly implements evaluation's argument-passing mechanism. The converse implication in the lemma also holds but we shall not need it and, as its proof requires the introduction of non-trivial formalism, we shall not pursue the matter.

3.3 Soundness and Error-Freeness

In analogy with standard evaluation, we can also show that stack-based evaluation is sound and correct in the limited sense of being error-free.

Lemma 20 (Soundness) *Let $s \in \{n, v\}$, see Figures 7 and 8.*

$$\begin{array}{l} \Gamma \mid \Delta \rhd\!\!> e : \tau \ \wedge \ \models^{\rhd\!\!>} \rho : \Gamma \ \wedge \ \models^{\rhd\!\!>} \delta : \Delta \ \wedge \ \mathrm{Seval}^s(e, \rho, \delta) = r \\ \Downarrow \\ \models^{\rhd\!\!>} r : \tau \end{array}$$

Proof See Appendix B.2. □

Theorem 21 (Error-Freeness) *Let $s \in \{n, v\}$, see Figures 7 and 8.*

$$\bot \mid \varepsilon \rhd\!\!> e : \tau \ \Rightarrow \ \mathrm{Seval}^s(e, \bot, \varepsilon) \neq \texttt{error}$$

Proof By Proposition 15 and Lemma 20. □

$$
\begin{array}{rcl}
\Pi \models^{\triangleright\!\!\gg} \rho \mathrel{\overline{:}} \Gamma & \Leftrightarrow^{\text{def}} & \left\{ \begin{array}{l} \mathrm{Dom}(\rho) = \mathrm{Dom}(\Gamma) \wedge \\ \forall x \in \mathrm{Dom}(\rho).\Pi \models^{\triangleright\!\!\gg} \rho(x) \mathrel{\overline{:}} \Gamma(x) \end{array} \right. \\
\Pi \models^{\triangleright\!\!\gg} b \mathrel{\underline{:}} \tau & \Leftrightarrow^{\text{def}} & \bot; \Pi \triangleright\!\!\gg b \mathrel{\underline{:}} \tau \\
\Pi \models^{\triangleright\!\!\gg} d \mathrel{\overline{:}} o & \Leftrightarrow^{\text{def}} & \Pi \models^{\triangleright\!\!\gg} d \mathrel{\underline{:}} o \\
\Pi \models^{\triangleright\!\!\gg} [e, \rho] \mathrel{\overline{:}} \tau & \Leftrightarrow^{\text{def}} & \exists \Gamma.\Pi \models^{\triangleright\!\!\gg} \rho \mathrel{\overline{:}} \Gamma \wedge \Gamma; \Pi \triangleright\!\!\gg e \mathrel{\overline{:}} \tau \\
\Pi \models^{\triangleright\!\!\gg} \langle x, e, \rho \rangle \mathrel{\overline{:}} \tau & \Leftrightarrow^{\text{def}} & \exists \Gamma.\Pi \models^{\triangleright\!\!\gg} \rho \mathrel{\overline{:}} \Gamma \wedge \Gamma; \Pi \triangleright\!\!\gg \overline{\lambda} x.e \mathrel{\overline{:}} \tau
\end{array}
$$

Figure 11: NbE value typing

Proposition 22 (Digest) *Well-typed stack-based evaluation will*

- *not yield* `error` *from any sub-computation,*
- *bind any encountered variable in the associated environment, and*
- *return with an empty stack, when terminating.*

Proof For the first two properties, juxtapose Theorem 21 with the rules that generate and propagate `error` in Figures 7 and 8. For the last property, observe that the non `error`-yielding axioms in Figures 7 and 8 specify an empty stack and that the other non-`error` rules merely yield as result something that was yielded further up. □

4 Normalisation by Evaluation

In this section, we briefly revisit Section 1.3 and [2] to state the relevant formal results for NbE in the style of the previous sections.

4.1 Type Theory

We first present the relevant notion of NbE value typing, where we note that free (underlined) *y*s are fully acceptable because, as *syntax*, they will be considered immutable at the operational, i.e., overlined, level.

Definition 23 *NbE value typing,* $\models^{\triangleright\!\!\gg}$*, is defined in Figure 11.*

Proposition 24 $\Pi \not\models^{\triangleright\!\!\gg} \mathtt{error} \mathrel{\underline{:}} \tau \wedge \Pi \not\models^{\triangleright\!\!\gg} \mathtt{error} \mathrel{\overline{:}} \tau$

Secondly, we show that also NbE typing is a conservative extension of standard simple typing.

Lemma 25 *See Figures 1 and 4.*

$$\Gamma \rhd e : \tau \quad \Leftrightarrow \quad \Gamma; \bot \triangleright\!\!\gg \ulcorner e \urcorner \mathrel{\overline{:}} \tau$$

Proof By straightforward rule inductions. □

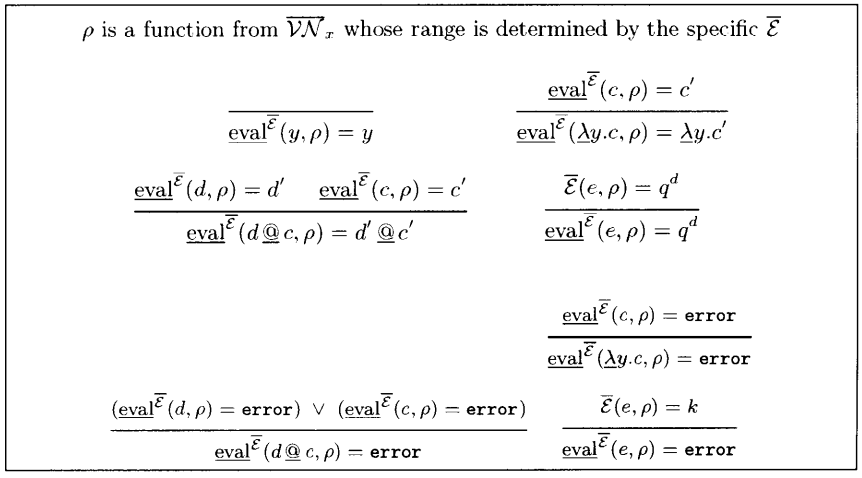


Figure 12: Big-step semantics for syntactic pass-through to semantic evaluation $\overline{\mathcal{E}}$; formally speaking, a renaming environment for ys is needed to avoid variable-capture (within ρ) in the abstraction rule

With this, we can show that NbE and standard value typing are equivalent.

Lemma 26

$$\begin{array}{rcll} (\models^{\triangleright} \rho : \Gamma & \Leftrightarrow & \bot \models^{\triangleright\!\!\triangleright} \ulcorner \rho \urcorner \mathbin{\dot{:}} \Gamma) & \wedge \\ (\models^{\triangleright} [e,\rho] : \tau & \Leftrightarrow & \bot \models^{\triangleright\!\!\triangleright} [\ulcorner e \urcorner, \ulcorner \rho \urcorner] \mathbin{\dot{:}} \tau) & \wedge \\ (\models^{\triangleright} \langle x,e,\rho \rangle : \tau & \Leftrightarrow & \bot \models^{\triangleright\!\!\triangleright} \langle x, \ulcorner e \urcorner, \ulcorner \rho \urcorner \rangle \mathbin{\dot{:}} \tau) & \end{array}$$

Proof By straightforward triple rule inductions, using Lemma 25. □

4.2 Operational Semantics

The operational semantics of NbE is two-sorted to reflect the two levels of terms we recounted in Section 1.3.[3] Evaluation of the underlined, or "syntactical", level is basically the identity function as syntax, naturally, is immutable as far as semantical evaluation is concerned. Figure 12 presents the details.

Proposition 27 (Well-definedness) $\underline{\mathrm{eval}}^{\overline{\mathcal{E}}}(-,-)$ *is functional, provided* $\overline{\mathcal{E}}$ *is.*

Proof The case-splitting is non-overlapping in the figure. □

[3]Strictly speaking, the terms are three sorted, which means that so is evaluation. However, we largely suppress the distinction between c and d and consider just one syntactic sort, b.

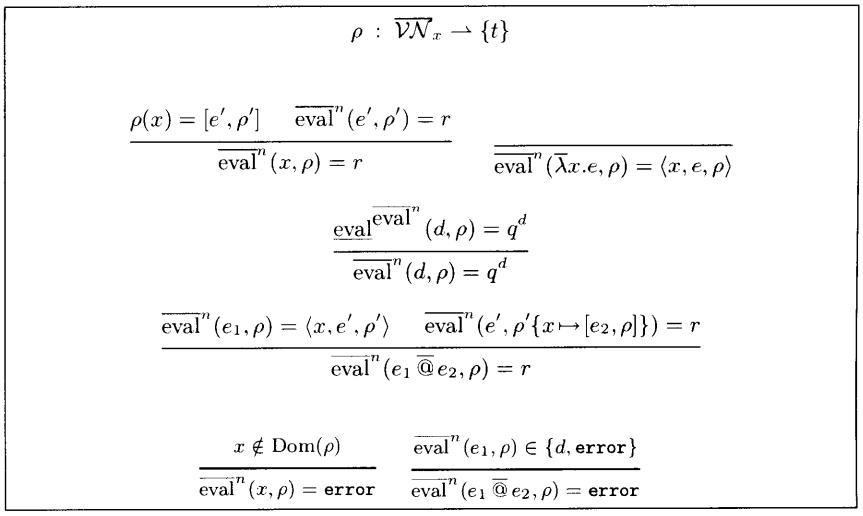


Figure 13: The $\mathrm{NbE}^{\mathrm{CbN}}$ big-step semantics (for semantics)

Proposition 28 (Quasi-totality) *If* $\underline{\mathrm{eval}}^{\overline{\mathcal{E}}}(b,\rho)$ *is undefined, some invocation of* $\overline{\mathcal{E}}$ *does not terminate.*

Proof The case-splitting is exhaustive in the figure and each recursive call of $\underline{\mathrm{eval}}^{\overline{\mathcal{E}}}$ is on a proper sub-term of the case term (i.e, $\underline{\mathrm{eval}}^{\overline{\mathcal{E}}}$ is defined structural-recursively, modulo $\overline{\mathcal{E}}$). □

Before leaving syntactic (pass-through) evaluation we note that, in order to prevent capture of free y's occurring in the term environments, ρ, we must also thread a renaming environment, σ, to be used as follows.

$$\frac{}{\underline{\mathrm{eval}}^{\overline{\mathcal{E}}}(y,\rho,\sigma)=\sigma(y)} \qquad \frac{\underline{\mathrm{eval}}^{\overline{\mathcal{E}}}(c,\rho,\sigma\{y\mapsto z\})=c' \quad z \text{ fresh}}{\underline{\mathrm{eval}}^{\overline{\mathcal{E}}}(\underline{\lambda}y.c,\rho,\sigma)=\underline{\lambda}z.c'} \tag{1}$$

We suppress the details here but stress that the use of a renaming environment *is* necessary[4] (and that, naturally, it is used in the accompanying ML implementations).

Unsurprisingly, the run-time values for the overlined NbE big-step semantics, see Figures 13 and 14, include underlined syntax as constants alongside closures:

$$\begin{array}{rcl} q & ::= & b \mid \texttt{error} \\ w & ::= & k \mid d \\ r & ::= & w \mid \texttt{error} \end{array}$$

[4]If renaming is not used, $\lambda x_1.x_1\,(\lambda x_2.x_1\,(\lambda x_3.x_i))$ will normalise-by-evaluation to a fixed term irrespective of whether x_i is x_2 or x_3 with thanks to A. Filinski.

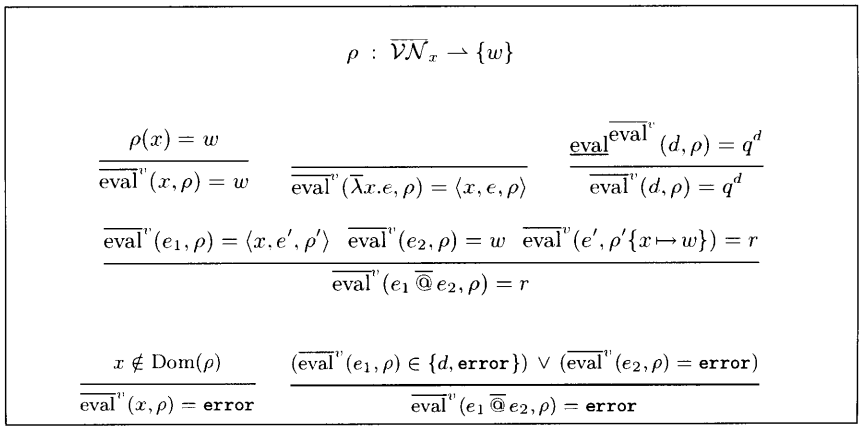


Figure 14: The NbE^{CbV} big-step semantics (for semantics)

As seen, we retain r for semantic run-time values and use q for syntactic run-time values and q^d when we wish to disambiguate b and stress that we mean d or `error`. For semantic run-time values, we use w to denote well-defined values, i.e., what the call-by-value big-step semantics stores in the environments. The call-by-name big-step semantics still stores thunks.

Proposition 29 (Well-definedness) $\overline{\text{eval}}^s(-,-)$ *is functional, for* $s \in \{n, v\}$.

Proof Further to Proposition 27, we note that the case-splitting is non-overlapping in each figure. □

Proposition 30 (Quasi-totality) *If, for* $s \in \{n, v\}$, $\overline{\text{eval}}^s(e,\rho)$ *is undefined, the evaluation of* e *relative to* ρ *does not terminate.*

Proof Further to Proposition 28, we note that the case-splitting is exhaustive in each figure. □

The final result of this section is semantical conservative extensivity (in the stronger form of equivalence) over standard evaluation.

Lemma 31 $\ulcorner \text{eval}^s(e,\rho) \urcorner = \overline{\text{eval}}^s(\ulcorner e \urcorner, \ulcorner \rho \urcorner)$, *for* $s \in \{n, v\}$.

Proof By two straightforward rule inductions, invoking the definition of $\ulcorner - \urcorner$ to show that the cases for $\overline{\text{eval}}^s$ that involve a d need not be considered. □

4.3 Soundness and Error-Freeness

In analogy with the earlier evaluation mechanisms, the results we have just presented allow us to state soundness and minimal correctness of NbE. We trust no further comments are necessary for the first three results.

Lemma 32 (Soundness) *Let* $s \in \{n, v\}$.

$$\left(\begin{array}{l} \Gamma; \Pi \rhd\!\!> b : \tau \ \wedge\ \Pi \models^{\rhd\!\!>} \rho : \Gamma \ \wedge\ \underline{\mathrm{eval}}^{\overline{\mathrm{eval}}^{s}}(b, \rho) = q \\ \Downarrow \\ \Pi \models^{\rhd\!\!>} q : \tau \end{array}\right)$$

$$\wedge$$

$$\left(\begin{array}{l} \Gamma; \Pi \rhd\!\!> e : \tau \ \wedge\ \Pi \models^{\rhd\!\!>} \rho : \Gamma \ \wedge\ \overline{\mathrm{eval}}^{s}(e, \rho) = r \\ \Downarrow \\ \Pi \models^{\rhd\!\!>} r : \tau \end{array}\right)$$

Proof See Appendix B.3. □

Theorem 33 (Error-Freeness) *Let* $s \in \{n, v\}$.

$$\bot; \Pi \rhd\!\!> b : \tau \ \Rightarrow\ \underline{\mathrm{eval}}^{\overline{\mathrm{eval}}^{s}}(b, \bot) \neq \texttt{error}$$

$$\wedge$$

$$\bot; \Pi \rhd\!\!> e : \tau \ \Rightarrow\ \overline{\mathrm{eval}}^{s}(e, \bot) \neq \texttt{error}$$

Proof By Proposition 24 and Lemma 32. □

Proposition 34 (Digest) *Well-typed NbE will*

- *yield (underlined) syntax from (underlined) syntax, when terminating,*
- *yield a closure when applied to the function position of a semantical application (if the call terminates) and, more generally,*
- *not yield* `error` *from any sub-computation, and, finally,*
- *bind any encountered variables in the associated environment.*

As promised, soundness can be used to establish even more interesting results than the above notion of correctness in the case of NbE. Specifically, it can used to establish that NbE eponymously performs *normalisation by evaluation.*

Theorem 35 (Normalisation by Evaluation) *Let* $s \in \{n, v\}$.

$$\begin{array}{l} \bot \rhd e : \tau \ \wedge\ \underline{\mathrm{eval}}^{\overline{\mathrm{eval}}^{s}}(\downarrow^{\tau} \ulcorner e \urcorner, \bot) = q \\ \Downarrow \\ q \in \llcorner \Lambda_{\tau}^{\mathrm{long}\ \beta(\eta)} \lrcorner \end{array}$$

Proof The result is a special case of Theorem 49, according to Lemma 45. □

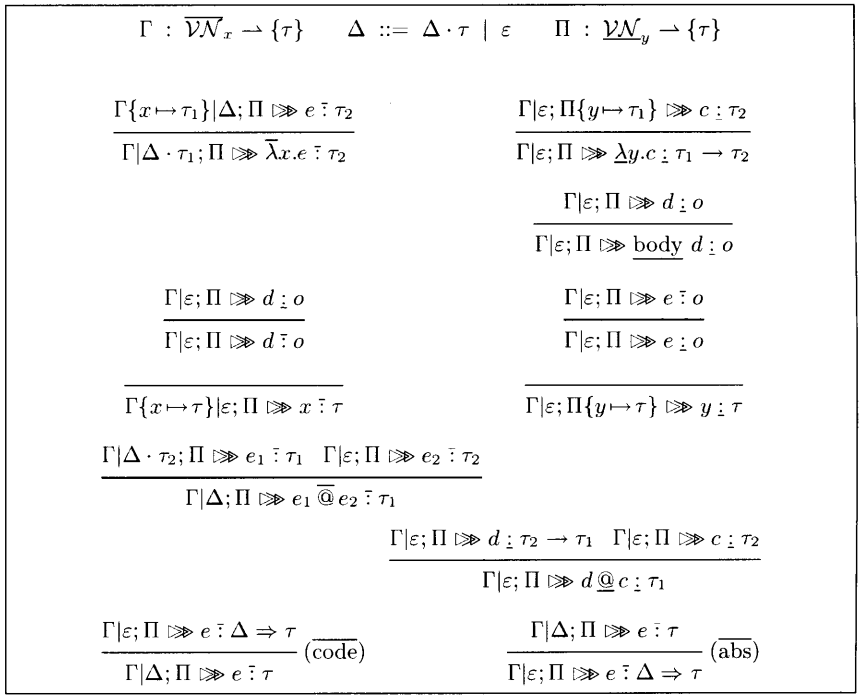


Figure 15: Simple, two-level types for NbSE

If we had presented a direct proof of Lemma 32, we could have noted that the $b \equiv e$-case shows that no overlined syntax can remain because a closure (as well as `error`) cannot be yielded due to the type constraint, which means that we have Theorem 35. The details of the argument are available in the combined proof of Lemma 46 and Theorem 49 in Appendix A.

5 Normalisation by Stack-Based Evaluation

Having given detailed, stand-alone accounts of NbE and stack-based evaluation, we will now combine them as Normalisation by Stack-based Evaluation, NbSE. This section presents their conservative-extension union. In the next section, we shall anticipate a later efficiency assessment by identifying and implementing what appears to be optimisations arising out of the complementary nature of "Nb" and "SE". The optimisations are not possible for either of the two in isolation and, as they break the naive conservative-extension property, they are addressed separately.

This section contains very little explanatory text because the technical format we employ is known by now and because, by construction, very few technical

$$
\begin{array}{rcl}
\Pi \models^{\rhd\!\!\gg} \rho : \Gamma & \Leftrightarrow^{\text{def}} & \left\{ \begin{array}{l} \mathrm{Dom}(\rho) = \mathrm{Dom}(\Gamma) \wedge \\ \forall x \in \mathrm{Dom}(\rho).\Pi \models^{\rhd\!\!\gg} \rho(x) : \Gamma(x) \end{array} \right. \\
\Pi \models^{\rhd\!\!\gg} \delta : \Delta & \Leftrightarrow^{\text{def}} & |\delta| = |\Delta| \wedge \forall i \in \{1, \ldots |\delta|\}.\Pi \models^{\rhd\!\!\gg} \delta.i : \Delta.i \\
\Pi \models^{\rhd\!\!\gg} b \mathbin{:} \tau & \Leftrightarrow^{\text{def}} & \bot|\varepsilon; \Pi \rhd\!\!\gg b \mathbin{:} \tau \\
\Pi \models^{\rhd\!\!\gg} d : o & \Leftrightarrow^{\text{def}} & \Pi \models^{\rhd\!\!\gg} d \mathbin{:} o \\
\Pi \models^{\rhd\!\!\gg} [e, \rho] : \tau & \Leftrightarrow^{\text{def}} & \exists \Gamma.\Pi \models^{\rhd\!\!\gg} \rho : \Gamma \wedge \Gamma|\varepsilon; \Pi \rhd\!\!\gg e : \tau \\
\Pi \models^{\rhd\!\!\gg} \langle x, e, \rho \rangle : \tau & \Leftrightarrow^{\text{def}} & \exists \Gamma.\Pi \models^{\rhd\!\!\gg} \rho : \Gamma \wedge \Gamma|\varepsilon; \Pi \rhd\!\!\gg \overline{\lambda} x.e : \tau
\end{array}
$$

Figure 16: NbSE value typing

subtleties are introduced relative to the previous two sections.

5.1 Type Theory

The new type system simply overlaps the two type systems just considered. A typing judgement for a "semantical" term, e, will thus be of the form $\Gamma|\Delta; \Pi \rhd\!\!\gg e : \tau$ and, for a "syntactical" term, b, $\Gamma|\varepsilon; \Pi \rhd\!\!\gg b \mathbin{:} \tau$ where Γ and Δ are an environment and a stack for the semantics, respectively, and Π is an environment for the syntax. Figure 15 gives the typing rules.

Lemma 36

NbSE/SE typing:

$$\Gamma \mid \Delta \rhd\!\!\gg e : \tau \quad \Leftrightarrow \quad \Gamma|\Delta; \bot \rhd\!\!\gg \ulcorner e \urcorner : \tau$$

NbSE/NbE typing:

$$\begin{array}{rcl} (\Gamma; \Pi \rhd\!\!\gg e : \tau & \Leftrightarrow & \Gamma|\varepsilon; \Pi \rhd\!\!\gg e : \tau) \quad \wedge \\ (\Gamma; \Pi \rhd\!\!\gg b : \tau & \Leftrightarrow & \Gamma|\varepsilon; \Pi \rhd\!\!\gg b \mathbin{:} \tau) \end{array}$$

Proof By straightforward rule inductions. □

In the NbSE/NbE case, the first property is a consequence of the following, more general equivalence result.

Lemma 37

$$\Gamma; \Pi \rhd\!\!\gg e : \Delta \Rightarrow \tau \;\Leftrightarrow\; \Gamma|\Delta; \Pi \rhd\!\!\gg e : \tau$$

Proof From Lemma 36, using the ($\overline{\text{code}}$) and ($\overline{\text{abs}}$) rules of Figure 15. □

As we have seen, the results above carry over to value typing.

Definition 38 *NbSE value typing, $\models^{\rhd\!\!\gg}$, is defined in Figure 16.*

Proposition 39 $\Pi \not\models^{\rhd\!\!\gg} \texttt{error} \mathbin{:} \tau \wedge \Pi \not\models^{\rhd\!\!\gg} \texttt{error} : \tau$

Lemma 40

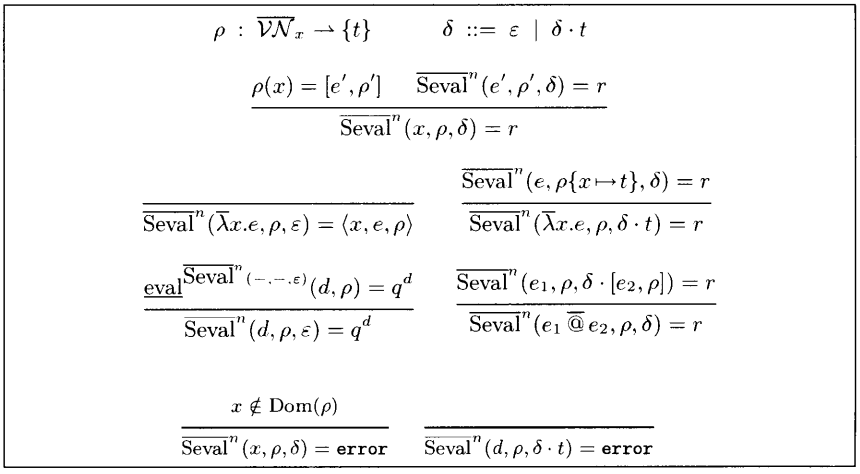


Figure 17: The $\mathrm{NbSE}^{\mathrm{CbN}}$ big-step semantics (for semantics)

NbSE/SE value typing:

$$\begin{array}{rcll}
(\models^{\triangleright} \rho : \Gamma & \Leftrightarrow & \bot \models^{\triangleright\!\!\triangleright} \ulcorner\rho\urcorner \mathbin{:} \Gamma) & \wedge \\
(\models^{\triangleright} \delta : \Delta & \Leftrightarrow & \bot \models^{\triangleright\!\!\triangleright} \ulcorner\delta\urcorner \mathbin{:} \Delta) & \wedge \\
(\models^{\triangleright} [e, \rho] : \tau & \Leftrightarrow & \bot \models^{\triangleright\!\!\triangleright} [\ulcorner e\urcorner, \ulcorner\rho\urcorner] \mathbin{:} \tau) & \wedge \\
(\models^{\triangleright} \langle x, e, \rho\rangle : \tau & \Leftrightarrow & \bot \models^{\triangleright\!\!\triangleright} \langle x, \ulcorner e\urcorner, \ulcorner\rho\urcorner\rangle \mathbin{:} \tau) &
\end{array}$$

NbSE/NbE value typing:

$$\begin{array}{rcll}
(\Pi \models^{\triangleright} \rho \mathbin{:} \Gamma & \Leftrightarrow & \Pi \models^{\triangleright\!\!\triangleright} \rho \mathbin{:} \Gamma) & \wedge \\
(\Pi \models^{\triangleright} b \mathbin{:} \tau & \Leftrightarrow & \Pi \models^{\triangleright\!\!\triangleright} b \mathbin{:} \tau) & \wedge \\
(\Pi \models^{\triangleright} d \mathbin{:} o & \Leftrightarrow & \Pi \models^{\triangleright\!\!\triangleright} d \mathbin{:} o) & \wedge \\
(\Pi \models^{\triangleright} [e, \rho] \mathbin{:} \tau & \Leftrightarrow & \Pi \models^{\triangleright\!\!\triangleright} [e, \rho] \mathbin{:} \tau) & \wedge \\
(\Pi \models^{\triangleright} \langle x, e, \rho\rangle \mathbin{:} \tau & \Leftrightarrow & \Pi \models^{\triangleright\!\!\triangleright} \langle x, e, \rho\rangle \mathbin{:} \tau) &
\end{array}$$

Proof By simultaneous rule inductions, using Lemma 36. □

5.2 Operational Semantics

The run-time values for evaluation in this system are as follows.

$$\begin{array}{rcl}
q & ::= & b \mid \texttt{error} \\
\\
w & ::= & k \mid d \\
r & ::= & w \mid \texttt{error}
\end{array}$$

Big-step semantics Refer to Figures 17 and 18 for the details and confer with Figures 7 and 8 for the originals.

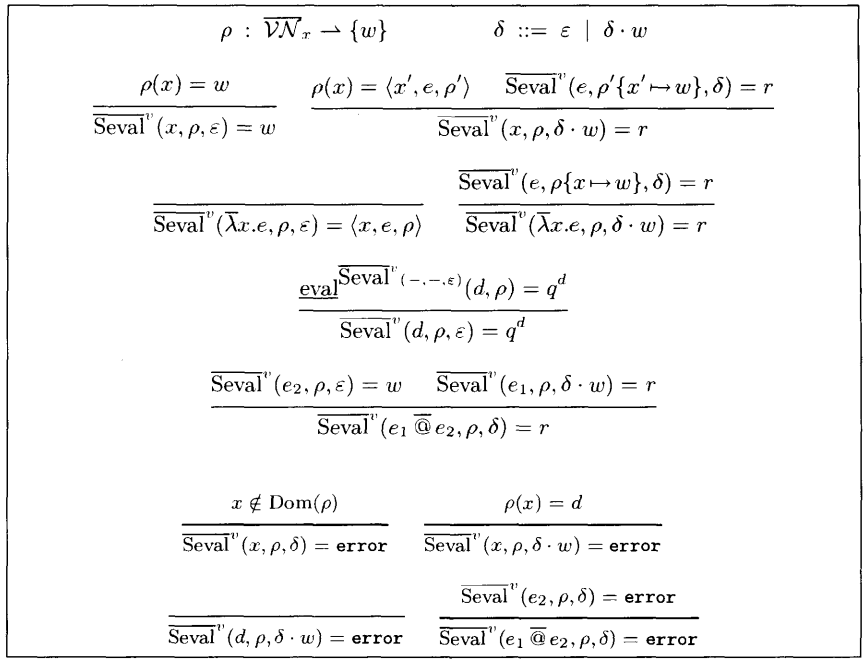


Figure 18: The NbSE$^{\mathrm{CbV}}$ big-step semantics (for semantics)

Proposition 41 (Well-definedness) $\overline{\mathrm{Seval}}^{s}(-,-,-)$ *is functional, for* $s \in \{n, v\}$.

Proof Further to Proposition 27, we note that the case-splitting is non-overlapping in each figure. □

Proposition 42 (Quasi-totality) *If, for* $s \in \{n, v\}$, $\overline{\mathrm{Seval}}^{s}(e, \rho, \delta)$ *is undefined, the evaluation of* e *relative to* ρ *and* δ *does not terminate.*

Proof Further to Proposition 28, we note that the case-splitting is exhaustive in each figure. □

Conservative Extensivity Like for stack-based evaluation proper, we shall need "continuation" lemmas for NbSE in order to prove conservative-extensivity relative to the corresponding stack-free notion of evaluation, viz. NbE.

Lemma 43 *Let* $s \in \{n, v\}$, *cf. Figures 2, 3, 7, and 8.*

$$\overline{\mathrm{Seval}}^{s}(e, \rho, \delta) = \mathtt{error} \Rightarrow \overline{\mathrm{Seval}}^{s}(e, \rho, \delta'\delta) = \mathtt{error}$$

Proof By a straightforward rule induction. □

Lemma 44 *Let $s \in \{n, v\}$ and let l^s be t in case $s = n$ and w in case $s = v$.*

$$\begin{array}{l}\overline{\mathrm{Seval}}^s(e, \rho, \delta) = \langle z, e_0, \rho_0 \rangle \ \wedge \ \overline{\mathrm{Seval}}^s(e_0, \rho_0\{z \mapsto l^s\}, \varepsilon) = r \\ \Downarrow \\ \overline{\mathrm{Seval}}^s(e, \rho, l^s\delta) = r\end{array}$$

Proof By a straightforward rule induction in $\overline{\mathrm{Seval}}^s(e, \rho, \delta)$. As it turns out, a double induction, involving also $\underline{\mathrm{eval}}^-$, is not necessary because the considered e cannot be a d (because a closure is yielded). □

Lemma 45 *Let $s \in \{n, v\}$.*

NbSE/SE:

$$\mathrm{Seval}^s(e, \rho, \delta) = r \quad \Rightarrow \quad \overline{\mathrm{Seval}}^s(\ulcorner e \urcorner, \ulcorner \rho \urcorner, \ulcorner \delta \urcorner) = \ulcorner r \urcorner$$

NbSE/NbE:

$$\begin{array}{rcll}(\underline{\mathrm{eval}}^{\overline{\mathrm{eval}}^s}(b, \rho) = q & \Rightarrow & \underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(b, \rho) = q) & \wedge \\ (\overline{\mathrm{eval}}^s(e, \rho) = r & \Rightarrow & \overline{\mathrm{Seval}}^s(e, \rho, \varepsilon) = r) & \end{array}$$

Proof By rule inductions. For the NbSE/NbE case, we proceed by a double rule induction, using Lemmas 43 and 44 for semantical application. □

5.3 Soundness, Error-Freeness, and NbSE

Lemma 46 (Soundness) *Let $s \in \{n, v\}$.*

$$\left(\begin{array}{l}\Gamma|\varepsilon; \Pi \Rrightarrow b : \tau \ \wedge \ \Pi \models^{\Rrightarrow} \rho : \Gamma \ \wedge \ \underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(b, \rho) = q \\ \Downarrow \\ \Pi \models^{\Rrightarrow} q : \tau\end{array}\right)$$

$$\wedge$$

$$\left(\begin{array}{l}\Gamma|\Delta; \Pi \Rrightarrow e : \tau \ \wedge \ \Pi \models^{\Rrightarrow} \rho : \Gamma \ \wedge \ \Pi \models^{\Rrightarrow} \delta : \Delta \ \wedge \ \overline{\mathrm{Seval}}^s(e, \rho, \delta) = r \\ \Downarrow \\ \Pi \models^{\Rrightarrow} r : \tau\end{array}\right)$$

Proof See Appendix A; the proof contains no surprises. □

Theorem 47 (Error-Freeness) *Let $s \in \{n, v\}$.*

$$\bot|\varepsilon; \Pi \Rrightarrow b : \tau \ \Rightarrow \ \underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(b, \bot) \neq \texttt{error}$$

$$\wedge$$

$$\bot|\varepsilon; \Pi \Rrightarrow e : \tau \ \Rightarrow \ \overline{\mathrm{Seval}}^s(e, \bot, \varepsilon) \neq \texttt{error}$$

Proof By Proposition 39 and Lemma 46. □

Proposition 48 (Digest) *Well-typed NbSE will*

- *never abandon a non-empty stack, even when returning,*
- *yield (underlined) syntax from (underlined) syntax, when terminating,*
- *bind any encountered variables in the associated environment, and*
- *not yield* `error` *from any sub-computation.*

Theorem 49 (Normalisation by Stack-Based Evaluation) *Let* $s \in \{n, v\}$.

$$\begin{array}{l} \bot \rhd e : \tau \ \wedge \ \underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^{s}(-,-,\varepsilon)}(\downarrow^{\tau} \ulcorner e \urcorner, \bot) = q \\ \Downarrow \\ q \in \llcorner \Lambda_{\tau}^{\mathrm{long}\ \beta(\eta)} \lrcorner \end{array}$$

Proof See Appendix A; the proof piggybacks the proof of Lemma 46. □

6 Closure-Freeness

With NbSE defined naively, we shall now optimise it to be closure-free in the hope that this will result in a low administrative overhead. We note that it is the combination of "Nb" and "SE" that enables the optimisation and that neither of "Nb" and "SE" can be made closure-free by themselves, at least not in the prescribed manner.

6.1 Call-by-Name without Closures

When using NbSE for normalisation, see Lemma 49, all overlined CbN evaluation will natively take place at ground type. To see this, observe that overlined evaluation can only be initiated through the underlined level, i.e., at ground type (and with an empty stack). In the case of an overlined application, CbN evaluation will immediately address the function position, which strictly speaking means at higher type. However, due to the presence of the stack, which is extended with a thunk for the argument, the overall type stays at ground type. In other words, we can adapt naive NbSE to only consider ground-type run-time states, see Figure 19, and, in the process, convert Figure 17's closure-building rule (abstraction, empty stack) into an `error`-production rule that we do not encounter with well-typed terms (at ground type).

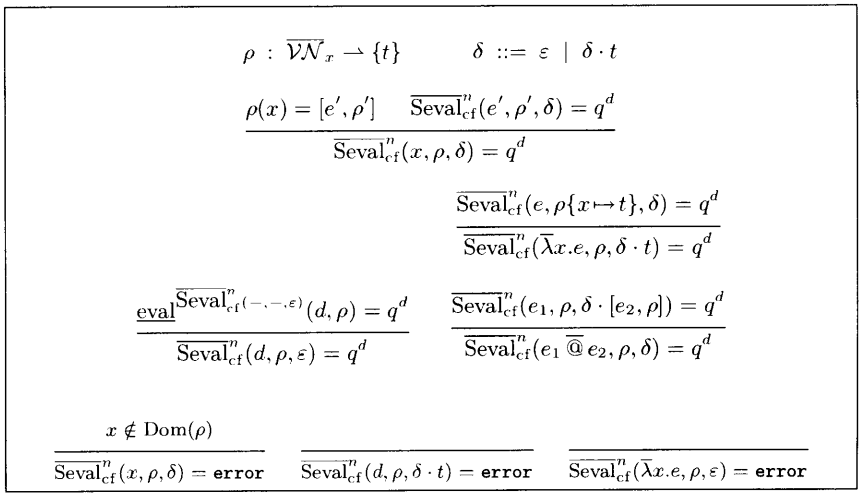


Figure 19: The (closure-free) $\mathrm{NbSE}^{\mathrm{CbN}}_{\mathrm{cf}}$ big-step semantics (for semantics)

Lemma 50 (Closure-Free Implementation Correctness)

$$\left(\begin{array}{l}\Gamma|\varepsilon;\Pi \rhd\!\!\gg b \mathrel{\dot{:}} \tau \ \wedge\ \Pi \models^{\rhd\!\!\gg} \rho \mathrel{\dot{:}} \Gamma \\ \Downarrow \\ \underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^{n}(-,-,\varepsilon)}(b,\rho) = \underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^{n}_{\mathrm{cf}}(-,-,\varepsilon)}(b,\rho)\end{array}\right)$$

$$\wedge$$

$$\left(\begin{array}{l}\Gamma|\Delta;\Pi \rhd\!\!\gg e \mathrel{\dot{:}} o \ \wedge\ \Pi \models^{\rhd\!\!\gg} \rho \mathrel{\dot{:}} \Gamma \ \wedge\ \Pi \models^{\rhd\!\!\gg} \delta \mathrel{\dot{:}} \Delta \\ \Downarrow \\ \overline{\mathrm{Seval}}^{n}(e,\rho,\delta) = \overline{\mathrm{Seval}}^{n}_{\mathrm{cf}}(e,\rho,\delta)\end{array}\right)$$

Proof By two (separate) double rule inductions for the left-to-right and right-to-left parts of the equalities. All cases are straightforward I.H.-applications, except for `error`-rules and the $\overline{\mathrm{Seval}}^{n}(\overline{\lambda}x.e,\rho,\varepsilon)$-case. These follow, instead, by an unlisted result stating that the scenarios in question violate the typing requirements. □

Theorem 51 (Closure-Free Call-by-Name NbSE)

$$\begin{array}{l}\bot \rhd e : \tau \ \wedge\ \underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^{n}_{\mathrm{cf}}(-,-,\varepsilon)}(\downarrow^{\tau} \ulcorner e \urcorner, \bot) = q \\ \Downarrow \\ q \in \llcorner \Lambda^{\mathrm{long}\ \beta(\eta)}_{\tau} \lrcorner\end{array}$$

Proof By Proposition 4, Corollary 13/Lemma 25, Lemma 36, Theorem 49, and Lemma 50. □

6.2 Call-by-Value without Closures

Unlike the situation with CbN, the CbV version of NbSE does not natively restrict computation to ground type. The difference lies in the evaluation of the argument position of an application for which no arguments are contained on the stack that is being considered, even if the argument term is of higher type. The relevant details can be found in the application rule in Figure 18, which uses an empty stack, ε, for the evaluation of e_2. The question we are interested in now is whether we can use a non-empty "speculative" stack instead of ε that allows us to stay at ground type from an overall perspective.

As it turns out, forcing potentially higher-typed semantical evaluation to be initiated at ground type is straightforward in the present set-up. Provided we know the type of the argument term, we can namely evaluate it through a syntactic "NbE interface": $\underline{\mathrm{eval}}^{-}(\downarrow^{\tau} e_2, \rho)$. Doing so will encapsulate e_2 by applications to reflected syntax, $\uparrow_{\tau_i} y_i$, until reaching ground type. This means that the actual overlined evaluation of e_2 we go on to perform will take place with a stack that keeps the overall type ground. The result of the computation is going to be a (fully) syntactic long $\beta(\eta)$-normal form, c. However,

- c cannot be used as the result because we need to apply e_1 to it (semantically).
- We cannot use $\ulcorner\!\|\, c \,\|\!\urcorner$ because c may contain free (syntactical) ys that we are not allowed to turn into free (semantical) xs.
- We cannot use $\uparrow_{\tau} c$ because the domain of $\uparrow_{\tau}$ is $\underline{D}_{\tau}$ and reflecting a c would ultimately break the syntactic restrictions on the underlined level and, with them, the results relying on those restrictions.

We will instead pursue a subtler approach that takes advantage of the analytic properties of long $\beta(\eta)$-normal forms: any and all ground type sub-terms occurs as a *body*.[5] We can, therefore, proceed as follows, see Figure 20: traverse the term, c, recursively turning the initial abstractions into semantical abstractions; upon reaching a body decide whether the head-variable is free in the overall term or not; if it is free, process the body as syntax; if it is not free, process the body as semantics.

Lemma 52 *Let* $b \in \underline{B}_{\tau}^{\overline{=}}$ *and let* Sem *be defined in Figure 20.*

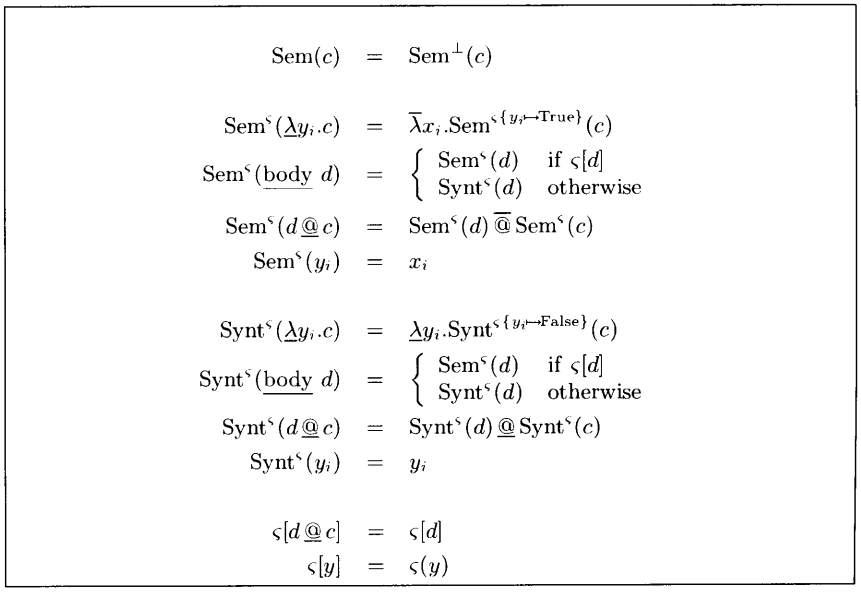
$$\bot|\varepsilon;\Pi \rhd\!\!\rhd b : \tau \quad \Rightarrow \quad \bot|\varepsilon;\Pi \rhd\!\!\rhd \mathrm{Sem}(b) \,\bar{:}\, \tau$$

Proof A straightforward double rule induction in Synt^{-} and Sem^{-}. □

Theorem 53 (Closure-Free Call-by-Value NbSE)

$$\begin{array}{l} \bot \rhd e : \tau \;\wedge\; \underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^{v}_{\mathrm{cf}(-,-,\varepsilon)}}(\downarrow^{\tau} \ulcorner e \urcorner, \bot) = q^{d} \\ \Downarrow \\ q^{d} \in \llcorner \Lambda_{\tau}^{\mathrm{long}\ \beta(\eta)} \lrcorner \end{array}$$

[5] Note that, e.g., $y_1 \,\underline{@}\, y_2$ actually amounts to $y_1 \,\underline{@}\, (\underline{\mathrm{body}}\ y_2)$, with y_2 (of ground type) trivially occurring fully-applied in a separate d.



$$
\begin{array}{rcl}
\mathrm{Sem}(c) & = & \mathrm{Sem}^{\perp}(c) \\
\\
\mathrm{Sem}^{\varsigma}(\underline{\lambda} y_i.c) & = & \overline{\lambda} x_i.\mathrm{Sem}^{\varsigma\{y_i \mapsto \mathrm{True}\}}(c) \\
\mathrm{Sem}^{\varsigma}(\underline{\mathrm{body}}\ d) & = & \begin{cases} \mathrm{Sem}^{\varsigma}(d) & \text{if } \varsigma[d] \\ \mathrm{Synt}^{\varsigma}(d) & \text{otherwise} \end{cases} \\
\mathrm{Sem}^{\varsigma}(d \mathbin{\underline{@}} c) & = & \mathrm{Sem}^{\varsigma}(d) \mathbin{\overline{@}} \mathrm{Sem}^{\varsigma}(c) \\
\mathrm{Sem}^{\varsigma}(y_i) & = & x_i \\
\\
\mathrm{Synt}^{\varsigma}(\underline{\lambda} y_i.c) & = & \underline{\lambda} y_i.\mathrm{Synt}^{\varsigma\{y_i \mapsto \mathrm{False}\}}(c) \\
\mathrm{Synt}^{\varsigma}(\underline{\mathrm{body}}\ d) & = & \begin{cases} \mathrm{Sem}^{\varsigma}(d) & \text{if } \varsigma[d] \\ \mathrm{Synt}^{\varsigma}(d) & \text{otherwise} \end{cases} \\
\mathrm{Synt}^{\varsigma}(d \mathbin{\underline{@}} c) & = & \mathrm{Synt}^{\varsigma}(d) \mathbin{\underline{@}} \mathrm{Synt}^{\varsigma}(c) \\
\mathrm{Synt}^{\varsigma}(y_i) & = & y_i \\
\\
\varsigma[d \mathbin{\underline{@}} c] & = & \varsigma[d] \\
\varsigma[y] & = & \varsigma(y)
\end{array}
$$

Figure 20: Converting open syntax to closed semantics

Proof The result follows by the obvious adaptation (and vast simplification of not considering closures, value typing, etc.) of the proof in Appendix A, where we use Lemma 52 for the required typing information in the altered application case, rather than a direct argument. □

7 Conclusion

We have shown that stack-based evaluation (SE) and normalisation by evaluation combine cleanly as NbSE. At the core of our presentation is a conservative-extension hierarchy reaching down to plain evaluation that allows us to prove just one type-soundness result and have it reflected down the hierarchy. In addition, we note that the NbE and NbSE theorems straightforwardly piggyback type soundness because of our two-level presentation of the considered language. Finally, we have shown how to implement NbSE without closures for a potential reduction in administrative overhead although the details remain to be investigated. The above works for both call-by-name (CbN) and call-by-value (CbV) and, moreover, closure-free CbV NbSE is more aggressive than usual CbV big-step semantics by fully normalising argument terms before passing them on.

Acknowledgement We thank A.Ohori for discussions about type soundness.

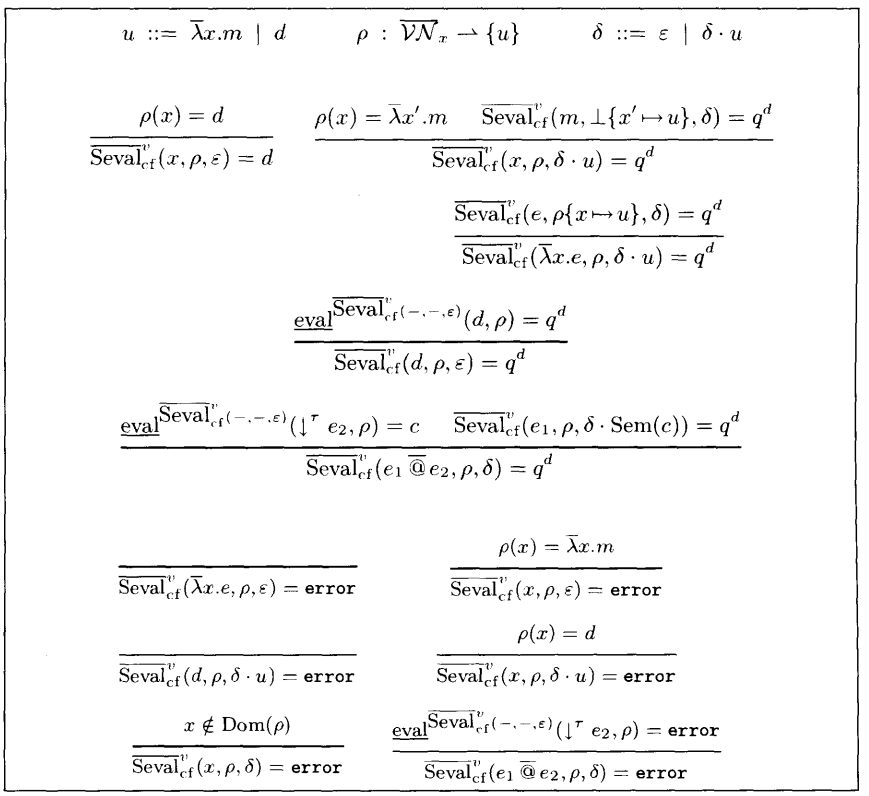


Figure 21: The (closure-free) $\mathrm{NbKM}^{\mathrm{CbV}}_{\mathrm{cf}}$ big-step semantics (for semantics)

A Combined Proof of Lemma 46, Theorem 49

The property we prove here relies on the obvious notion of NbE value-typing of stacks, defined as follows — it is readily seen to coincide with $\Pi \models^{\rhd\!\!\rhd} \delta : \Delta$.

$$\Pi \models^{\rhd} \delta : \Delta \;\Leftrightarrow^{\mathrm{def}}\; |\delta| = |\Delta| \wedge \forall i \in \{1, \ldots |\delta|\}.\Pi \models^{\rhd} \delta.i : \Delta.i$$

More precisely, we prove the following, where i) quantifications and conjunctions are in the places they need to be for the proof to work and ii) further to Lemmas 36, 37, and 40, we use $\rhd$ in place of $\rhd\!\!\rhd$ because it follows the term

structure closer, thereby making a number of required arguments more direct.

$$\forall b, e, \rho, \delta, q, r\,.$$

$$(\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^{s}(-,-,\varepsilon)}(b,\rho) = q) \;\wedge\; (\overline{\mathrm{Seval}}^{s}(e,\rho,\delta) = r)$$

$$\Downarrow$$

$$\left(\begin{array}{c} \forall \Gamma, \Pi, \tau\,.\; \left(\begin{array}{l} \Gamma; \Pi \rhd\!\!> b : \tau \;\wedge\; \Pi \models^{\rhd\!\!>} \rho : \Gamma \\ \Downarrow \\ \Pi \models^{\rhd\!\!>} q : \tau \;\wedge\; q \in \underline{B}^{=}_{\tau} \end{array}\right) \\ \\ \wedge \\ \\ \forall \Gamma, \Pi, \Delta, \tau\,.\; \left(\begin{array}{l} \Gamma; \Pi \rhd\!\!> e : \Delta \Rightarrow \tau \;\wedge\; \Pi \models^{\rhd\!\!>} \rho : \Gamma \;\wedge\; \Pi \models^{\rhd\!\!>} \delta : \Delta \\ \Downarrow \\ \Pi \models^{\rhd\!\!>} r : \tau \;\wedge\; (\forall b\,.\, r = b \;\Rightarrow\; r \in \underline{B}^{=}_{\tau}) \end{array}\right) \end{array}\right)$$

We say 'first conclusion' to mean the first conjunct in the inner-most conclusions and likewise for second. We note that in order to establish the second conlusions in the CbV case, value-typing of environments must be extended with the clause $(\forall b\,.\, w = b \;\Rightarrow\; w \in \underline{B}^{=}_{\tau})$, for each value, w, in the environments. The property is respected because we only put yielded values in the environment and they enjoy the required property by I.H. — we suppress the details.

Proof We consider arbitrary values for the outer-most universal quantifications, assume the statements about evaluation, and proceed by a double rule induction in $\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^{s}(-,-,\varepsilon)}$ and $\overline{\mathrm{Seval}}^{n}/\overline{\mathrm{Seval}}^{v}$. The typical case proceeds thus

1. split the conjunction (inside one set of parentheses) and, for each conjunct,
2. consider arbitrary values for the universally quantified $\Gamma, \Pi, (\Delta,)\tau$,
3. assume the premises of the implication (inside two sets of parentheses),
4. take note of the relevant sub-cases and their I.H.; one of the sub-cases for the non-`error` case for $d \mathbin{\underline{@}} c$, e.g., is $\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^{s}(-,-,\varepsilon)}(d,\rho) = d'$ and I.H.:

$$\forall \Gamma, \Pi, \tau\,.\; \left(\begin{array}{l} \Gamma; \Pi \rhd\!\!> d : \tau \;\wedge\; \Pi \models^{\rhd\!\!>} \rho : \Gamma \\ \Downarrow \\ \Pi \models^{\rhd\!\!>} d' : \tau \;\wedge\; d' \in \underline{B}^{=}_{\tau} \end{array}\right)$$

5. show the premises of the I.H. for the particular Γ', Π', τ' of the sub-case,
6. discharge the I.H., thereby concluding, e.g., $\Pi' \models^{\rhd\!\!>} d' : \tau' \;\wedge\; d' \in \underline{B}^{=}_{\tau'}$,
7. use this to conclude, e.g., $(\Pi \models^{\rhd\!\!>} d' \mathbin{\underline{@}} c' : \tau) \wedge (d' \mathbin{\underline{@}} c' \in \underline{B}^{=}_{\tau})$, as required.

Steps 1–4 are left implicit. We present the cases in linearised form: the first typing-implication (conjunct inside one set of parentheses) is established by the first evaluation assumption and likewise for the second typing-implication and evaluation assumption. **First, we consider the cases for** $\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}$.

Case $\dfrac{}{\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(y,\rho) = y}$:

The (assumed) term-typing premise implies (by an unstated lemma) that $\Pi(y) = \tau$ and therefore that $\bot; \Pi \rhd\!\!> y : \tau$, as required for the first conclusion. The second conclusion holds by definition.

Case $\dfrac{\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(c,\rho) = c'}{\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(\underline{\lambda} y.c,\rho) = \underline{\lambda} y.c'}$:

$\dfrac{\Gamma; \Pi\{y \mapsto \tau_1\} \rhd\!\!> c : \tau_2}{\Gamma; \Pi \rhd\!\!> \underline{\lambda} y.c : \tau_1 \to \tau_2}$ is the only rule that could have resulted in the (assumed) term-typing premise (further to an unstated lemma that abstractions cannot be of ground type, o). We thus have $\Pi\{y \mapsto \tau_1\} \models^{\rhd\!\!>} c' : \tau_2$ and $c' \in \underline{B}^{\overline{\ }}_{\tau_2}$ by I.H. (and two un-stated weakening lemmas implying that $\Pi\{y \mapsto \tau_1\}$ can be used instead of Π in both premises). This implies that we have $\bot; \Pi\{y \mapsto \tau_1\} \rhd\!\!> c' : \tau_2$ and both conclusions are now simple.

Case $\dfrac{\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(d,\rho) = d' \quad \underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(c,\rho) = c'}{\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(d \mathbin{\underline{@}} c,\rho) = d' \mathbin{\underline{@}} c'}$:

$\dfrac{\Gamma; \Pi \rhd\!\!> d : \tau_2 \to \tau_1 \quad \Gamma; \Pi \rhd\!\!> c : \tau_2}{\Gamma; \Pi \rhd\!\!> d \mathbin{\underline{@}} c : \tau_1}$ and $\dfrac{\Gamma; \Pi \rhd\!\!> e \mathbin{\bar{:}} o}{\Gamma; \Pi \rhd\!\!> e : o}$ are the only rules that could have resulted in the (assumed) term-typing premise. In the latter case, only $\dfrac{\Gamma; \Pi \rhd\!\!> d' : o}{\Gamma; \Pi \rhd\!\!> d' \mathbin{\bar{:}} o}$ could have preceded the considered rule and only the former typing rule could have preceded that (because of the terms' inductive structure). In other words, we have $\dfrac{\Gamma; \Pi \rhd\!\!> d : \tau_2 \to \tau_1 \quad \Gamma; \Pi \rhd\!\!> c : \tau_2}{\Gamma; \Pi \rhd\!\!> d \mathbin{\underline{@}} c : \tau_1}$.
The I.H. for evaluation of d and c can therefore have their implications discharged and the desired conclusions are straightforward.

Case $\dfrac{\overline{\mathrm{Seval}}^s(e,\rho,\varepsilon) = q^d}{\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(e,\rho) = q^d}$:

$\dfrac{\Gamma; \Pi \rhd\!\!> e \mathbin{\bar{:}} o}{\Gamma; \Pi \rhd\!\!> e : o}$ is the only rule that could have resulted in the (assumed) term-typing premise and we are straightforwardly done by I.H. (involving $\overline{\mathrm{Seval}}^s$) (and an unstated lemma that all r such that $\Pi \models^{\rhd\!\!>} r \mathbin{\bar{:}} o$ are a b, i.e., that closures, k, cannot be of ground type, o). For discharging the implication of the I.H., note that stack-value-typing holds trivially for ε.

Case $\dfrac{\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(c,\rho) = \mathtt{error}}{\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(\underline{\lambda} y.c,\rho) = \mathtt{error}}$:

Part of the argument in the non-`error` case for $\underline{\lambda} y.c$, including discharge of the I.H., and Proposition 24 show that the assumptions are contradictory.

Case $\dfrac{(\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(d,\rho) = \mathtt{error}) \ \vee \ (\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(c,\rho) = \mathtt{error})}{\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(d \mathbin{\underline{@}} c,\rho) = \mathtt{error}}$:

Part of the argument in the non-`error` case for $d \mathbin{\underline{@}} c$, including discharge of the I.H., and Proposition 24 show that the assumptions are contradictory.

Case $\dfrac{\overline{\mathrm{Seval}}^s(e,\rho,\varepsilon) = k}{\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s(-,-,\varepsilon)}(e,\rho) = \mathtt{error}}$:

Part of the argument in the non-`error` case for e, including the unstated lemma, shows that the assumptions are contradictory.

Next, we consider the cases for $\overline{\mathrm{Seval}}^n$.

Case $\dfrac{\rho(x) = [e',\rho'] \quad \overline{\mathrm{Seval}}^n(e',\rho',\delta) = r}{\overline{\mathrm{Seval}}^n(x,\rho,\delta) = r}$:

The I.H. can be discharged by the definitional meaning of $\Pi \models^{\rhd\!\rhd} [e',\rho'] \mathbin{\bar{:}} \tau$ and we are done because r is also the finally yielded value.

Case $\dfrac{}{\overline{\mathrm{Seval}}^n(\overline{\lambda} x.e,\rho,\varepsilon) = \langle x,e,\rho\rangle}$:

The first conclusion holds by assumption, the second holds trivially.

Case $\dfrac{\overline{\mathrm{Seval}}^n(e,\rho\{x \mapsto t\},\delta) = r}{\overline{\mathrm{Seval}}^n(\overline{\lambda} x.e,\rho,\delta \cdot t) = r}$:

$\dfrac{\Gamma\{x \mapsto \tau_1\}; \Pi \rhd\!\rhd e \mathbin{\bar{:}} \tau_2}{\Gamma; \Pi \rhd\!\rhd \overline{\lambda} x.e \mathbin{\bar{:}} \tau_1 \to \tau_2}$ is the only rule that could have resulted in the (assumed) term-typing premise and by definition of value-typing stacks, we must have some Δ' such that $\Delta = \Delta' \cdot \tau_1$. We can directly discharge the I.H. with $\Gamma\{x \mapsto \tau_1\}, \Pi, \Delta', \tau_2$ and we are done because r is also the finally yielded value.

Case $\dfrac{\underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^n(-,-,\varepsilon)}(d,\rho) = q^d}{\overline{\mathrm{Seval}}^n(d,\rho,\varepsilon) = q^d}$:

$\dfrac{\Gamma; \Pi \rhd\!\rhd d \mathbin{\underline{:}} o}{\Gamma; \Pi \rhd\!\rhd d \mathbin{\bar{:}} o}$ is the only rule that could have resulted in the (assumed) term-typing premise and the I.H. can be discharged by the assumptions of the case and we are done because q^d is also the finally yielded value and because overlined and underlined value-typing coincide at ground type, o.

Case $\dfrac{\overline{\text{Seval}}^{n}(e_1, \rho, \delta \cdot [e_2, \rho]) = r}{\overline{\text{Seval}}^{n}(e_1 \,\overline{@}\, e_2, \rho, \delta) = r}$:

$\dfrac{\Gamma; \Pi \rhd\!\!> e_1 \mathbin{:} \tau_2 \to \tau_1 \quad \Gamma; \Pi \rhd\!\!> e_2 \mathbin{:} \tau_2}{\Gamma; \Pi \rhd\!\!> e_1 \,\overline{@}\, e_2 \mathbin{:} \tau_1}$ is the only rule that could have resulted in the (assumed) term-typing premise ($\dfrac{\Gamma; \Pi \rhd\!\!> d \mathbin{:} o}{\Gamma; \Pi \rhd\!\!> d \mathbin{:} o}$ cannot be used because $e_1 \,\overline{@}\, e_2$ cannot be a d). Rearranging the assumptions allows is to discharge the I.H. and we are done because r is also the finally yielded value.

Case $\dfrac{x \notin \text{Dom}(\rho)}{\overline{\text{Seval}}^{n}(x, \rho, \delta) = \texttt{error}}$:

$\dfrac{}{\Gamma\{x \mapsto \tau\}; \Pi \rhd\!\!> x \mathbin{:} \tau}$ is the only rule that could have resulted in the (assumed) term-typing premise. The facts that $x \in \text{Dom}(\Gamma\{x \mapsto \tau\})$ and $x \notin \text{Dom}(\rho)$ are inconsistent with the assumption that $\Pi \models^{\rhd\!\!>} \rho \mathbin{:} \Gamma$ and we are trivially done.

Case $\dfrac{}{\overline{\text{Seval}}^{n}(d, \rho, \delta \cdot t) = \texttt{error}}$:

$\dfrac{\Gamma; \Pi \rhd\!\!> d \mathbin{:} o}{\Gamma; \Pi \rhd\!\!> d \mathbin{:} o}$ is the only rule that could have resulted in the (assumed) term-typing premise. The fact that we are considering a non-empty stack at ground type, o, is inconsistent because of the assumption that reads $\Pi \models^{\rhd\!\!>} (\delta \cdot t) \mathbin{:} \varepsilon$ and we are trivially done.

Finally, we consider the cases for $\overline{\text{Seval}}^{v}$ that differ from $\overline{\text{Seval}}^{n}$.

Case $\dfrac{\rho(x) = w}{\overline{\text{Seval}}^{v}(x, \rho, \varepsilon) = w}$:

We are straightforwardly done by definition of the assumed $\Pi \models^{\rhd\!\!>} \rho \mathbin{:} \Gamma$ (with the extension to the relation discussed above).

Case $\dfrac{\rho(x) = \langle x', e, \rho' \rangle \quad \overline{\text{Seval}}^{v}(e, \rho'\{x' \mapsto w\}, \delta) = r}{\overline{\text{Seval}}^{v}(x, \rho, \delta \cdot w) = r}$:

The I.H. can be discharged by definition of the assumed $\Pi \models^{\rhd\!\!>} \rho \mathbin{:} \Gamma$, in the particular case of x, with the obvious instantiations of the universal quantifications, and we are done because r is also the finally yielded value.

Case $\dfrac{\overline{\text{Seval}}^{v}(e_2, \rho, \varepsilon) = w \quad \overline{\text{Seval}}^{v}(e_1, \rho, \delta \cdot w) = r}{\overline{\text{Seval}}^{v}(e_1 \,\overline{@}\, e_2, \rho, \delta) = r}$:

Similar to the case for $\dfrac{\overline{\text{Seval}}^{n}(e_1, \rho, \delta \cdot [e_2, \rho]) = r}{\overline{\text{Seval}}^{n}(e_1 \mathbin{\overline{@}} e_2, \rho, \delta) = r}$, except that the considered typing rule is used to first discharge the I.H. involving e_2, with the conclusion of that used to help discharge the I.H. for e_1; we are then done because r is also the finally yielded value.

Case $\dfrac{\rho(x) = d}{\overline{\text{Seval}}^{v}(x, \rho, \delta \cdot w) = \texttt{error}}$ **:**

By construction of the type system and definition of the assumed $\Pi \models^{\triangleright\!\!\triangleright} \rho \mathbin{:} \Gamma$, the type in question is ground, o. We are therefore trivially done by a similar argument to that used or $\dfrac{}{\overline{\text{Seval}}^{n}(d, \rho, \delta \cdot t) = \texttt{error}}$.

Case $\dfrac{\overline{\text{Seval}}^{v}(e_2, \rho, \delta) = \texttt{error}}{\overline{\text{Seval}}^{v}(e_1 \mathbin{\overline{@}} e_2, \rho, \delta) = \texttt{error}}$ **:**

Further to the case for $\dfrac{\overline{\text{Seval}}^{v}(e_2, \rho, \varepsilon) = w \qquad \overline{\text{Seval}}^{v}(e_1, \rho, \delta \cdot w) = r}{\overline{\text{Seval}}^{v}(e_1 \mathbin{\overline{@}} e_2, \rho, \delta) = r}$, we are trivially because the assumptions are contradictory according to Proposition 24. □

B Deriving Results from NbSE

B.1 Evaluation

We derive Lemma 7 from both Lemmas 32 and 20, independently. We first assume the premises of the former lemma:

$$\Gamma \rhd e : \tau \wedge \models^{\triangleright} \rho : \Gamma \wedge \text{eval}^{s}(e, \rho) = r$$

B.1.1 From Normalisation by Evaluation

By Lemmas 25, 26, and 31, we have

$$\Gamma; \bot \rhd\!\!\rhd \ulcorner e \urcorner \mathbin{:} \tau \wedge \bot \models^{\triangleright\!\!\triangleright} \ulcorner \rho \urcorner \mathbin{:} \Gamma \wedge \overline{\text{eval}}^{s}(\ulcorner e \urcorner, \ulcorner \rho \urcorner) = \ulcorner r \urcorner$$

From Lemma 32, we can therefore conclude

$$\bot \models^{\triangleright\!\!\triangleright} \ulcorner r \urcorner \mathbin{:} \tau$$

This implies that $r \neq \texttt{error}$ and we are done by Lemma 26.

B.1.2 From Stack-based Evaluation

From Corollary 13 and Lemmas 16 and 19, we have

$$\Gamma \mid \varepsilon \rhd\!\!> e : \tau \wedge \models^{\rhd} \rho : \Gamma \wedge \mathrm{Seval}^s(e, \rho, \varepsilon) = r$$

As we trivially have $\models^{\rhd} \varepsilon : \varepsilon$, we can apply Lemma 20 to conclude

$$\models^{\rhd} r : \tau$$

This implies that $r \neq \texttt{error}$ and we are done by Lemma 16.

B.2 Stack-based Evaluation

We derive Lemma 20 from Lemma 46. First, we assume the premises of the former lemma.

$$\Gamma \mid \Delta \rhd\!\!> e : \tau \wedge \models^{\rhd} \rho : \Gamma \wedge \models^{\rhd} \delta : \Delta \wedge \mathrm{Seval}^s(e, \rho, \delta) = r$$

By Lemmas 36, 40, and 45, we therefore have

$$\begin{aligned}&\Gamma|\Delta; \bot \rhd\!\!\rhd\!\!> \ulcorner e \urcorner : \tau \wedge \bot \models^{\rhd\!\!\rhd\!\!>} \ulcorner \rho \urcorner : \Gamma \\ &\wedge \bot \models^{\rhd\!\!\rhd\!\!>} \ulcorner \delta \urcorner : \Delta \wedge \overline{\mathrm{Seval}}^s(\ulcorner e \urcorner, \ulcorner \rho \urcorner, \ulcorner \delta \urcorner) = \ulcorner r \urcorner\end{aligned}$$

By Lemma 46, we can conclude

$$\bot \models^{\rhd\!\!\rhd\!\!>} \ulcorner r \urcorner : \tau$$

This implies that $r \neq \texttt{error}$ and we are done by Lemma 40.

B.3 Normalisation by Evaluation

We derive Lemma 32 from Lemma 46.

B.3.1 Syntactic Soundness

First, we assume the premises pertaining to syntax:

$$\Gamma; \Pi \rhd\!\!> b : \tau \wedge \Pi \models^{\rhd} \rho : \Gamma \wedge \underline{\mathrm{eval}}^{\overline{\mathrm{eval}}^s}(b, \rho) = q$$

By Lemmas 36, 40, and 45, we therefore have

$$\Gamma|\varepsilon; \Pi \rhd\!\!\rhd\!\!> b : \tau \wedge \Pi \models^{\rhd\!\!\rhd\!\!>} \rho : \Gamma \wedge \underline{\mathrm{eval}}^{\overline{\mathrm{Seval}}^s}(b, \rho) = q$$

By Lemma 46, we can conclude

$$\Pi \models^{\rhd\!\!\rhd\!\!>} q : \tau$$

This implies that $q \neq \texttt{error}$ and we are done by Lemma 40.

B.3.2 Semantic Soundness

Next, we assume the premises pertaining to semantics:

$$\Gamma; \Pi \rhd\!\!\gg e \mathrel{:} \tau \ \wedge\ \Pi \models^{\rhd\!\!\gg} \rho \mathrel{:} \Gamma \ \wedge\ \overline{\mathrm{eval}}^{s}(e, \rho) = r$$

By Lemmas 36, 40, and 45, we therefore have

$$\Gamma|\varepsilon; \Pi \rhd\!\!\gg e \mathrel{:} \tau \ \wedge\ \Pi \models^{\rhd\!\!\gg} \rho \mathrel{:} \Gamma \ \wedge\ \overline{\mathrm{Seval}}^{s}(\epsilon, \rho, \varepsilon) = r$$

As we trivially have $\Pi \models^{\rhd\!\!\gg} \varepsilon \mathrel{:} \varepsilon$, we can apply Lemma 46 to conclude

$$\Pi \models^{\rhd\!\!\gg} r \mathrel{:} \tau$$

This implies that $r \neq \texttt{error}$ and we are done by Lemma 40.

References

[1] Gérard Huet. Résolution d'équations dans les langages d'ordre 1, 2, ..., ω. Thèse d'État, Université de Paris VII, Paris, France, 1976.

[2] René Vestergaard. The simple type theory of normalisation by evaluation. *Electronic Notes in Theoretical Computer Science*, 57, 2001.

[3] René Vestergaard. *The Primitive Proof Theory of the λ-Calculus.* PhD thesis, School of Mathematical and Computer Sciences, Heriot-Watt University, 2003.