Strong Normalization for Simply Typed Call-by-Push-Value

Jonathan Chan

1 Introduction

The goal of this project is to mechanize a proof of strong normalization for call-by-push-value (CBPV). It follows work by Forster, Schäfer, Spies, and Stark (TODO: cite) on mechanizing the metatheory of CBPV in Rocq, but instead adapts the proof technique from POPLMark Reloaded (TODO: cite).

Both proofs, and mine, follow the same sequence of steps: we define a logical relation that semantically interpret types as sets of terms, show that these sets are backward closed by reduction and that they contain only strongly normalizing terms, prove a fundamental soundness theorem that places well typed terms in the semantic interpretations of their types, and conclude that well typed terms must be strongly normalizing. The difference lies in *how strong normalization* (and reduction) is defined. The Rocq mechanization uses a traditional definition of strong normalization as an accessibility relation, while POPLMark Reloaded and this project use an inductive characterization of strong normalization, then prove it sound with respect to the traditional definition.

In contrast to both prior works, which use Rocq, Agda, or Beluga, this project is mechanized in Lean. A secondary purpose of this project is to assess the capabilities of Lean for PL metatheory, especially when making heavy use of mutually defined inductive types. I have implemented a mutual induction tactic for Lean (TODO: link), and the mutual values and computations of CBPV, combined with the even more complex mutual inductives from POPLMark Reloaded, make it an attractive test case to test the tactic's robustness.

This report is divided in roughly the same shape as the proof development. Section 2 introduces the syntax and the typing rules for CBPV (`Syntax.lean`, `Typing.lean`). Then the inductive characterization of strong normalization is defined in Section 3 (`NormalInd.lean`), followed by the logical relation in Section 4 (`OpenSemantics.lean`). The central theorem of strong normalization is proven in Section 5 as a corollary of the fundamental theorem for the logical relation (`Soundness.lean`, `Normalization.lean`). To ensure that the inductive characterization is correct, Section 6 shows that it implies the traditional definition of strong normalization (`NormalAcc.lean`). This latter definition and proof depends on small-step reduction on open terms, whose properties are omitted here (`Reduction.lean`). In general, I gloss over details about substitution and reduction, since I'm interested in presenting the structure of the strong normalization proof and not the minutiae of syntax and binding. Finally, I discuss the merits of this proof technique and the challenges posed by using Lean in Section 7.

2 Syntax and Typing

$$
\begin{array}{rll}
A & ::= \mathsf{Unit} \mid A_1 + A_2 \mid \mathsf{U}\ B & \textit{value types} \\
B & ::= A \to B \mid \mathsf{F}\ A & \textit{computation types} \\
v, w & ::= x \mid \mathsf{unit} \mid \mathsf{inl}\ v \mid \mathsf{inr}\ v \mid \mathsf{thunk}\ m & \textit{values} \\
m, n & ::= \mathsf{force}\ v \mid \lambda x.\ m \mid m\ v \mid \mathsf{return}\ v & \textit{computations} \\
 & \quad \mid \mathsf{let}\ x \leftarrow m\ \mathsf{in}\ n & \\
 & \quad \mid \mathsf{case}\ v\ \mathsf{of}\ \{\mathsf{inl}\ y \Rightarrow m\ ;\ \mathsf{inr}\ z \Rightarrow n\} & \\
\Gamma & ::= \cdot \mid \Gamma, x : A & \textit{typing contexts} \\
\sigma & ::= \cdot \mid \sigma, x \mapsto v & \textit{simultaneous substitutions}
\end{array}
$$

The grammar of the CBPV for this project is given above. Terms and their types are divided between *values* and *computations*. Values are variables, unit, injections into a sum type, or a thunked computation, while computations are forcing thunks, functions, applications, returning values, binding returned values, and case analysis on sums. There are only value variables and no computation variables, since they represent terms that require no more work to be done on them ready to be substituted in, and typing contexts similarly only contain value types. I use t to refer to terms that may be either values or computations.

Although the syntax is presented nominally, the mechanization uses an unscoped de Bruijn indexed representation of variables, along with simultaneous substitutions σ mapping variables to values, with $\cdot$ as the identity substitution. Applying a substitution is represented as $v[\sigma]$ and $m[\sigma]$, and implemented in terms of renamings, which are mappings from variables to other variables.

This is not the usual complete CBPV language, since it's missing both value tuples and computation tuples. I exclude them because they are syntactically not much more interesting than returns, whose eliminator is shaped like pattern matching on a singleton value tuple, and than thunks, whose eliminator is shaped like projection on a singleton computation tuple. In contrast, I do include sums, which represent the only way a computation can branch.

The typing rules below are standard for the values and computations included. The judgements for values and computations are defined mutually, just as are the types and the terms.

$\boxed{\Gamma \vdash v : A}$ *value typing*

$$\textsc{T-Var}\ \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \textsc{T-Unit}\ \frac{}{\Gamma \vdash \mathsf{unit} : \mathsf{Unit}} \qquad \textsc{T-Inl}\ \frac{\Gamma \vdash v : A_1}{\Gamma \vdash \mathsf{inl}\ v : A_1 + A_2} \qquad \textsc{T-Inr}\ \frac{\Gamma \vdash v : A_2}{\Gamma \vdash \mathsf{inr}\ v : A_1 + A_2} \qquad \textsc{T-Thunk}\ \frac{\Gamma \vdash m : B}{\Gamma \vdash \mathsf{thunk}\ m : \mathsf{U}\ B}$$

$\boxed{\Gamma \vdash m : B}$ *computation typing*

$$\textsc{T-Force}\ \frac{\Gamma \vdash v : \mathsf{U}\ B}{\Gamma \vdash \mathsf{force}\ v : B} \qquad \textsc{T-Lam}\ \frac{\Gamma, x : A \vdash m : B}{\Gamma \vdash \lambda x.\, m : A \to B} \qquad \textsc{T-App}\ \frac{\Gamma \vdash m : A \to B \qquad \Gamma \vdash v : A}{\Gamma \vdash m\ v : B} \qquad \textsc{T-Ret}\ \frac{\Gamma \vdash v : A}{\Gamma \vdash \mathsf{return}\ v : \mathsf{F}\ A}$$

$$\textsc{T-Let}\ \frac{\Gamma \vdash m : \mathsf{F}\ A \qquad \Gamma, x : A \vdash n : B}{\Gamma \vdash \mathsf{let}\ x \leftarrow m\ \mathsf{in}\ n : B} \qquad \textsc{T-Case}\ \frac{\begin{array}{c}\Gamma \vdash v : A_1 + A_2 \\ \Gamma, y : A_1 \vdash m : B \\ \Gamma, z : A_2 \vdash n : B\end{array}}{\Gamma \vdash \mathsf{case}\ v\ \mathsf{of}\ \{\mathsf{inl}\ y \Rightarrow m\ ;\ \mathsf{inr}\ z \Rightarrow n\} : B}$$

3 Strong Normalization as an Inductive Definition

The idea behind the inductive characterization of strong normalization is to describe case by case when a term is strongly normal, rather than showing *a posteriori* which terms are strongly normal. This is done in conjunction with defining strongly neutral terms, which are blocked from β-reduction, and strong head reduction, which expands the set from normal forms to terms which must reduce to normal forms.

The general recipe for defining these for CBPV is as follows:

- The only strongly neutral values are variables.
- Strongly neutral computations are eliminators whose head positions are strongly neutral, while all other subterms are strongly normal.
- Strongly normal values are constructors whose subterms are strongly normal, or strongly neutral values, *i.e.* variables.
- Strongly normal computations are constructors whose subterms are strongly normal, or strongly neutral computations, or computations which reduce to strongly normal computations (backward closure).
- Strong head reduction consists of β reductions for all eliminators around the corresponding constructors, and congruent reductions in head positions.

Additionally, strong reduction requires premises asserting that the subterms that may "disappear" after reduction be strongly normal so that backward closure actually closes over strongly normal terms. These subterms are the values that get substituted into a computation, which may disappear if the computation never actually uses the binding, as well as computation branches not taken.

Because we're dealing with CBPV, values have no β reductions, so there's no need for head reduction of values as there are no heads. Furthermore, there is only one strongly neutral value, so we inline the definition as a variable where needed, but also write it as $v \in \mathsf{SNe}$ for symmetry as appropriate. Otherwise, the remaining four judgements are defined mutually below.

$\boxed{m \in \mathsf{SNe}}$ *strongly neutral computations*

$$\textsc{SNe-Force}\ \frac{}{\mathsf{force}\ x \in \mathsf{SNe}} \qquad \textsc{SNe-App}\ \frac{m \in \mathsf{SNe} \qquad v \in \mathsf{SN}}{m\ v \in \mathsf{SNe}} \qquad \textsc{SNe-Let}\ \frac{m \in \mathsf{SNe} \qquad n \in \mathsf{SN}}{\mathsf{let}\ x \leftarrow m\ \mathsf{in}\ n \in \mathsf{SNe}}$$

$$\textsc{SNe-Case}\ \frac{m \in \mathsf{SN} \qquad n \in \mathsf{SN}}{\mathsf{case}\ x\ \mathsf{of}\ \{\mathsf{inl}\ y \Rightarrow m\ ;\ \mathsf{inr}\ z \Rightarrow n\} \in \mathsf{SNe}}$$

$\boxed{v \in \mathsf{SN}}$ *strongly normal values*

$$\textsc{SN-Var}\ \frac{}{x \in \mathsf{SN}} \qquad \textsc{SN-Unit}\ \frac{}{\mathsf{unit} \in \mathsf{SN}} \qquad \textsc{SN-Inl}\ \frac{v \in \mathsf{SN}}{\mathsf{inl}\ v \in \mathsf{SN}} \qquad \textsc{SN-Inr}\ \frac{v \in \mathsf{SN}}{\mathsf{inr}\ v \in \mathsf{SN}} \qquad \textsc{SN-Thunk}\ \frac{m \in \mathsf{SN}}{\mathsf{thunk}\ m \in \mathsf{SN}}$$

$\boxed{m \in \mathsf{SN}}$ *strongly normal computations*

$$\textsc{SN-Lam}\ \frac{m \in \mathsf{SN}}{\lambda x.\, m \in \mathsf{SN}} \qquad \textsc{SN-Ret}\ \frac{v \in \mathsf{SN}}{\mathsf{return}\ v \in \mathsf{SN}} \qquad \textsc{SN-SNe}\ \frac{m \in \mathsf{SNe}}{m \in \mathsf{SN}} \qquad \textsc{SN-Red}\ \frac{m \rightsquigarrow n \qquad n \in \mathsf{SN}}{m \in \mathsf{SN}}$$

$\boxed{m \rightsquigarrow n}$ *strong head reduction*

$$\textsc{SR-Thunk}\ \frac{}{\mathsf{force}\ (\mathsf{thunk}\ m) \rightsquigarrow m} \qquad \textsc{SR-Lam}\ \frac{v \in \mathsf{SN}}{(\lambda x.\, m)\ v \rightsquigarrow m[x \mapsto v]} \qquad \textsc{SR-Ret}\ \frac{v \in \mathsf{SN}}{\mathsf{let}\ x \leftarrow (\mathsf{return}\ v)\ \mathsf{in}\ m \rightsquigarrow m[x \mapsto v]}$$

$$\textsc{SR-Inl}\ \frac{v \in \mathsf{SN} \qquad n \in \mathsf{SN}}{\mathsf{case}\ (\mathsf{inl}\ v)\ \mathsf{of}\ \{\mathsf{inl}\ y \Rightarrow m\ ;\ \mathsf{inr}\ z \Rightarrow n\} \rightsquigarrow m[y \mapsto v]} \qquad \textsc{SR-Inr}\ \frac{v \in \mathsf{SN} \qquad m \in \mathsf{SN}}{\mathsf{case}\ (\mathsf{inr}\ v)\ \mathsf{of}\ \{\mathsf{inl}\ y \Rightarrow m\ ;\ \mathsf{inr}\ z \Rightarrow n\} \rightsquigarrow n[z \mapsto v]}$$

$$\textsc{SR-App}\ \frac{m \rightsquigarrow n}{m\ v \rightsquigarrow n\ v} \qquad \textsc{SR-Let}\ \frac{m \rightsquigarrow m'}{\mathsf{let}\ x \leftarrow m\ \mathsf{in}\ n \rightsquigarrow \mathsf{let}\ x \leftarrow m'\ \mathsf{in}\ n}$$

$\boxed{m \rightsquigarrow^* n}$ *reflexive, transitive closure of head reduction*

$$\textsc{SRs-Refl}\ \frac{}{m \rightsquigarrow^* m} \qquad \textsc{SRs-Trans}\ \frac{m \rightsquigarrow m' \qquad m' \rightsquigarrow^* n}{m \rightsquigarrow^* n}$$

We also need the reflexive, transitive closure of head reduction, defined as a separate inductive above. Now we show a few simple lemmas about it, along with an inversion lemma for forcing (other inversion lemmas hold, but this is the only one we need). I present them below as judgement pseudorules using a dashed line to indicate that they are admissible rules. These are all proven either by construction or by induction on the first premise.

$$\textsc{SRs-Once}\ \frac{m \rightsquigarrow n}{m \rightsquigarrow^* n} \qquad \textsc{SRs-Trans'}\ \frac{m \rightsquigarrow^* m' \qquad m' \rightsquigarrow^* n}{m \rightsquigarrow^* n} \qquad \textsc{SRs-App}\ \frac{m \rightsquigarrow^* n}{m\ v \rightsquigarrow^* n\ v} \qquad \textsc{SRs-Let}\ \frac{m \rightsquigarrow^* m'}{\mathsf{let}\ x \leftarrow m\ \mathsf{in}\ n \rightsquigarrow^* \mathsf{let}\ x \leftarrow m'\ \mathsf{in}\ n}$$

$$\textsc{SN-Reds}\ \frac{m \rightsquigarrow^* n \qquad n \in \mathsf{SN}}{m \in \mathsf{SN}} \qquad \textsc{SN-Force-inv}\ \frac{\mathsf{force}\ v \in \mathsf{SN}}{v \in \mathsf{SN}}$$

The most important property of strongly normal terms that we need is antirenaming, which states that undoing a renaming does not change the term's normalization or reduction behaviour. A crucial property that follows is extensionality of applications, which is an inversion lemma specifically when the application argument is a variable.

Lemma 1 (Antirenaming)**.** *Let σ be a renaming, i.e. a substitution mapping variables to variables. Then the following hold:*

1. *If $m[\sigma] \in \mathsf{SNe}$ then $m \in \mathsf{SNe}$.*
2. *If $v[\sigma] \in \mathsf{SN}$ then $v \in \mathsf{SN}$.*
3. *If $m[\sigma] \in \mathsf{SN}$ then $m \in \mathsf{SN}$.*
4. *If $m[\sigma] \rightsquigarrow n$ then there exists some n' such that $n = n'[\sigma]$ and $m \rightsquigarrow n'$.*

Proof. By mutual induction on the four derivations of $m[\sigma] \in \mathsf{SNe}$, $v[\sigma] \in \mathsf{SN}$, $m[\sigma] \in \mathsf{SN}$, and $m[\sigma] \rightsquigarrow n$. □

Corollary 2 (Extensionality)**.** *If $m\ x \in \mathsf{SN}$ then $m \in \mathsf{SN}$.*

Proof. By induction on the derivation of strongly normal values. The possible cases are SN-SNe and SN-Red. Destructing the premise of the former gives $m \in \mathsf{SNe}$, and we conclude using rule SN-SNe again. In the latter case, we have $m\ x \rightsquigarrow n$ and $n \in \mathsf{SN}$; destructing on the reduction yields the possible cases rules SR-Lam and SR-App. In the first case, we have $m[x \mapsto v] \in \mathsf{SN}$, so the goal holds by Antirenaming. In the second case, we have $m \rightsquigarrow n$ and $n\ x \in \mathsf{SN}$, so by the induction hypothesis, we have $n \in \mathsf{SN}$, and we conclude using rule SN-Red. □

4 Logical Relation on Open Terms

The next step is to define a logical relation that semantically interprets types as sets of open terms. The key property we need from these sets is that they contain only strongly normal terms. Because we are working with open terms to prove normalization and not just termination of evaluation of closed terms, we need to consider variables and strongly neutral terms. Having no other information about them other than their strong neutrality, we require that the interpretation sets always contain all strongly neutral terms.

The logical relation on simple types can be defined by induction on the structure of the type. However, I want to maximize the amount of mutual inductives used in this project, so we instead define the logical relation as an inductive binary relation between the type and the set of terms of its interpretation, denoted as $[\![A]\!] \searrow P$ and $[\![B]\!] \searrow P$ below. I use set builder notation to define the sets and set membership, but they are implemented in the mechanization as functions that take terms and return propositions and as function applications. Here, the conjunctions ($\wedge$), disjunctions ($\vee$), equalities ($=$), implications ($\Rightarrow$), and universal ($\forall$) and existential ($\exists$) quantifiers are part of the metatheory, not part of the object language.

$\boxed{[\![A]\!] \searrow P}$ *semantic interpretation of value types*

$$\text{LR-Unit}\ \frac{}{[\![\mathsf{Unit}]\!] \searrow \{v \mid v \in \mathsf{SNe} \vee v = \mathsf{unit}\}} \qquad \text{LR-U}\ \frac{[\![B]\!] \searrow P}{[\![\mathsf{U}\ B]\!] \searrow \{v \mid \mathsf{force}\ v \in P\}}$$

$$\text{LR-Sum}\ \frac{[\![A_1]\!] \searrow P \qquad [\![A_2]\!] \searrow Q}{[\![A_1 + A_2]\!] \searrow \{v \mid v \in \mathsf{SNe} \vee (\exists w.\ v = \mathsf{inl}\ w \wedge w \in P) \vee (\exists w.\ v = \mathsf{inr}\ w \wedge w \in Q)\}}$$

$\boxed{[\![B]\!] \searrow P}$ *semantic interpretation of computation types*

$$\text{LR-F}\ \frac{[\![A]\!] \searrow P}{[\![\mathsf{F}\ A]\!] \searrow \{m \mid (\exists n.\ m \rightsquigarrow^* n \wedge n \in \mathsf{SNe}) \vee (\exists v.\ m \rightsquigarrow^* \mathsf{return}\ v \wedge v \in P)\}}$$

$$\text{LR-Arr}\ \frac{[\![A]\!] \searrow P \qquad [\![B]\!] \searrow Q}{[\![A \to B]\!] \searrow \{m \mid \forall v.\ v \in P \Rightarrow m\ v \in Q\}}$$

The terms in the interpretation of a type can be characterized in two different ways: by what constructor of that type they reduce to, or by how they act when eliminated by an eliminator for that type. For the former, we need to explicitly include the possibility of the term being strongly neutral. I chose the following characterizations because they seemed the simplest to me, but the alternate choices likely work as well.

- Values in the interpretation of the unit type are either variables or the unit value.
- Values are in the interpretation of the $\mathsf{U}\ B$ type if they can be forced to computations in the interpretation of the B type.
- Values in the interpretation of the sum type $A_1 + A_2$ are either variables, left injections whose values are in the interpretation of A_1, or right injections whose values are in that of A_2.
- Computations in the interpretation of the $\mathsf{F}\ A$ type reduce to either a neutral computation or a return whose value is in the interpretation of A.
- Computations are in the interpretation of the function type $A \to B$ if applying them to values in the interpretation of A yields computations in the interpretation of B.

By this description, it sounds like the logical relation can be presented directly as a relation between a type and a term; this presentation is given in Figure 1. Unfortunately, this is not well defined, since the logical relation appears in a negative position (to the left of an implication) in the premise of rule LRPP-Arr. The alternate presentation can be interpreted as a function match on the term and the type and returning the conjunction of its premises, but I wanted to use a mutually defined inductive definition.

$\boxed{v \in [\![A]\!]}$ $\boxed{m \in [\![B]\!]}$

$$\text{LR'-Unit-var}\ \frac{}{x \in [\![\mathsf{Unit}]\!]} \qquad \text{LR'-Unit-unit}\ \frac{}{\mathsf{unit} \in [\![\mathsf{Unit}]\!]} \qquad \text{LR'-Sum-var}\ \frac{}{x \in [\![A_1 + A_2]\!]}$$

$$\text{LR'-Sum-inl}\ \frac{v \in [\![A_1]\!]}{\mathsf{inl}\ v \in [\![A_1 + A_2]\!]} \qquad \text{LR'-Sum-inr}\ \frac{v \in [\![A_2]\!]}{\mathsf{inr}\ v \in [\![A_1 + A_2]\!]}$$

$$\text{LR'-Force}\ \frac{\mathsf{force}\ v \in [\![B]\!]}{v \in [\![\mathsf{U}\ B]\!]} \qquad \text{LR'-F-var}\ \frac{m \rightsquigarrow^* n \qquad n \in \mathsf{SNe}}{m \in [\![\mathsf{F}\ A]\!]} \qquad \text{LR'-F-ret}\ \frac{m \rightsquigarrow^* \mathsf{return}\ v \qquad v \in [\![A]\!]}{m \in [\![\mathsf{F}\ A]\!]}$$

$$\text{LR'-Arr}\ \frac{\forall v.\ v \in [\![A]\!] \Rightarrow m\ v \in [\![B]\!]}{m \in [\![A \to B]\!]}$$

Figure 1: Alternate presentation of the logical relation

Using the inductive presentation of the logical relation, there are three easy properties to show: interpretability, which states that all types have an interpretation; determinism, which states that the interpretation indeed behaves like a function from types to sets of terms; and backward closure, which states that

the interpretations of computation types are backward closed under multi-step head reduction. The last property is why rule LRPP-F-VAR needs to include computations that *reduce to* strongly neutral terms or returns, not merely ones that are such terms.

Lemma 3 (Interpretability)**.**
1. *Given A, there exists P such that $[\![A]\!] \searrow P$.*
2. *Given B, there exists P such that $[\![B]\!] \searrow P$.*

Proof. By mutual induction on the types A and B. □

Lemma 4 (Determinism)**.**
1. *If $[\![A]\!] \searrow P$ and $[\![A]\!] \searrow Q$ then $P = Q$.*
2. *If $[\![B]\!] \searrow P$ and $[\![B]\!] \searrow Q$ then $P = Q$.*

Proof. By mutual induction on the first derivations of the logical relation, then by cases on the second derivations. □

Lemma 5 (Backward closure)**.** *Given $[\![B]\!] \searrow P$ and $m \rightsquigarrow^* n$, if $n \in P$ then $m \in P$.*

Proof. By induction on the derivation of the logical relation. □

The final key property is adequacy, which states that the interpretations of types must contain all strongly neutral terms and must contain only strongly normal terms. Such sets are said to be *reducibility candidates*.

Definition 1 (Reducibility candidates)**.** A reducibility candidate is a set of terms P where, given a term t, if $t \in \mathsf{SNe}$ then $t \in P$, and if $t \in P$ then $t \in \mathsf{SN}$.

Lemma 6 (Adequacy)**.**
1. *If $[\![A]\!] \searrow P$ then P is a reducibility candidate.*
2. *If $[\![B]\!] \searrow P$ then P is a reducibility candidate.*

Proof. By mutual induction on the derivations of the logical relation. Rule SN-FORCE-INV is used in the S-U case, while rule SN-REDS is used in the rule S-F case. In the S-ARR case on $A \to B$, where $[\![A]\!] \searrow P$ and $[\![B]\!] \searrow Q$, to show the second part of its interpretation being a reducibility candidate, we are given m such that for every $v \in P$, $m\ v \in Q$, and the goal is to show that $m \in \mathsf{SN}$. By the induction hypotheses, picking an arbitrary variable x, we have that $x \in P$ (since it is neutral) and that $m\ x \in \mathsf{SN}$. Then the goal holds by Extensionality. □

5 Semantic Typing and the Fundamental Theorem

Now that we know that the interpretations contain only strongly normal terms, our goal is to show that well typed terms inhabit the interpretations of their types. We first define what it means for a substitution to be semantically well formed with respect to a context.

Definition 2 (Semantic well-formedness of substitutions)**.** A substitution σ is well formed with respect to a context Γ, written $\Gamma \vDash \sigma$, if for every $x : A \in \Gamma$ and $[\![A]\!] \searrow P$, we have $x[\sigma] \in P$. In short, the substitution maps variables in the context to values in the interpretations of their types.

These judgements can be built up inductively, as demonstrated by the below admissible rules, which are proven by cases.

$\boxed{\Gamma \vDash \sigma}$ *admissible semantic substitution well-formedness*

$$\frac{}{\Gamma \vDash \cdot}\ \text{S-NIL} \qquad \frac{\Gamma \vDash \sigma \qquad [\![A]\!] \searrow P \qquad v \in P}{\Gamma, x : A \vDash \sigma, x \mapsto v}\ \text{S-CONS}$$

Then we can define semantic typing in terms of the logical relation, using semantic well formedness of substitutions to handle the context.

Definition 3 (Semantic typing)**.**

1. v semantically has type A under context Γ, written $\Gamma \vDash v : A$, if for every σ such that $\Gamma \vDash \sigma$, there exists an interpretation $[\![A]\!] \searrow P$ such that $v[\sigma] \in P$.
2. m semantically has type B under context Γ, written $\Gamma \vDash m : B$, if for every σ such that $\Gamma \vDash \sigma$, there exists an interpretation $[\![B]\!] \searrow P$ such that $m[\sigma] \in P$.

Semantic typing follows exactly the same shape of rules as syntactic typing, so I present them here as admissible rules. All the hard work happens in these lemmas; the fundamental theorem of soundness of syntactic typing with respect to semantic typing then follows directly. Normalization holds as a corollary.

$\boxed{\Gamma \vDash v : A}$ *admissible semantic value typing*

$$\textsc{S-Var}\ \frac{x : A \in \Gamma}{\Gamma \vDash x : A} \qquad \textsc{S-Unit}\ \frac{}{\Gamma \vDash \mathsf{unit} : \mathsf{Unit}} \qquad \textsc{S-Inl}\ \frac{\Gamma \vDash v : A_1}{\Gamma \vDash \mathsf{inl}\ v : A_1 + A_2} \qquad \textsc{S-Inr}\ \frac{\Gamma \vDash v : A_2}{\Gamma \vDash \mathsf{inr}\ v : A_1 + A_2} \qquad \textsc{S-Thunk}\ \frac{\Gamma \vDash m : B}{\Gamma \vDash \mathsf{thunk}\ m : \mathsf{U}\ B}$$

$\boxed{\Gamma \vDash m : B}$ *admissible semantic computation typing*

$$\textsc{S-Force}\ \frac{\Gamma \vDash v : \mathsf{U}\ B}{\Gamma \vDash \mathsf{force}\ v : B} \qquad \textsc{S-Arr}\ \frac{\Gamma, x : A \vDash m : B}{\Gamma \vDash \lambda x.\, m : A \to B} \qquad \textsc{S-App}\ \frac{\Gamma \vDash m : A \to B \qquad \Gamma \vDash v : A}{\Gamma \vDash m\ v : B} \qquad \textsc{S-Ret}\ \frac{\Gamma \vDash v : A}{\Gamma \vDash \mathsf{return}\ v : \mathsf{F}\ A}$$

$$\textsc{S-Let}\ \frac{\Gamma \vDash m : \mathsf{F}\ A \qquad \Gamma, x : A \vDash n : B}{\Gamma \vDash \mathsf{let}\ x \leftarrow m\ \mathsf{in}\ n : B} \qquad \textsc{S-Case}\ \frac{\Gamma \vDash v : A_1 + A_2 \qquad \Gamma, y : A_1 \vDash m : B \qquad \Gamma, z : A_2 \vDash n : B}{\Gamma \vDash \mathsf{case}\ v\ \mathsf{of}\ \{\mathsf{inl}\ y \Rightarrow m\ ;\ \mathsf{inr}\ z \Rightarrow n\} : B}$$

Proof. By construction using prior lemmas, in particular rule SRs-Let in case S-Let; Interpretability in cases S-Var, S-Inl, S-Inr, and S-Lam; Determinism in cases S-Lam and S-App; Backward closure in cases S-Thunk, S-Lam, S-Let, and S-Case; and Adequacy in cases S-Lam, S-Let, and S-Case. □

Theorem 7 (Soundness)**.**

1. *If* $\Gamma \vdash v : A$ *then* $\Gamma \vDash v : A$.
2. *If* $\Gamma \vdash m : B$ *then* $\Gamma \vDash m : B$.

Proof. By mutual induction on the syntactic typing derivations, using the admissible semantic typing rules. □

Corollary 8 (Normalization)**.**

1. *If* $\Gamma \vdash v : A$ *then* $v \in \mathsf{SN}$.
2. *If* $\Gamma \vdash m : B$ *then* $m \in \mathsf{SN}$.

Proof. By Soundness, using the identity substitution and rule S-Nil, the well typed terms inhabit the semantic interpretations of their types. Then by Adequacy, they are also strongly normalizing. □

6 Strong Normalization as an Accessibility Relation

How do we know that our definition of strongly normal terms is correct? What are the properties that define its correctness? It seems to be insufficient to talk about strongly neutral and normal terms alone.

For instance, we could add another rule asserting that $m \in \mathsf{SN}$ if $m \rightsquigarrow m$, which doesn't appear to violate any of the existing properties we require. Then $(\lambda x.\,(\mathsf{force}\ x)\ x)\ (\mathsf{thunk}\ (\lambda x.\,(\mathsf{force}\ x)\ x))$, which reduces to itself, would be considered a "strongly normal" term, and typing rules that can type this term (perhaps with a recursive type) would still be sound with respect to our logical relation.

To try to rule out this kind of mistake in the definition, we might want to prove that strongly normal terms never loop, *i.e.* $m \in \mathsf{SN} \wedge m \rightsquigarrow^* m$ is impossible. However, this does not cover diverging terms that grow forever. Furthermore, $m \rightsquigarrow n$ isn't really a full reduction relation on its own, since it's defined mutually with strongly normal terms, and it doesn't describe reducing anywhere other than in the head position.

The solution in POPLMark Reloaded is to prove that SN is sound with respect to a traditional presentation of strongly normal terms based on full small-step reduction $v \rightsquigarrow w$, $m \rightsquigarrow n$ of values and computations. I omit here the definitions of these reductions, but they encompass all β-reduction rules and are congruent over all subterms, including under binders. Traditional strongly normal terms, written $v \in \mathsf{sn}$, $m \in \mathsf{sn}$, are accessibility relations with respect to small-step reduction. Terms are inductively defined as strongly normal if they step to strongly normal terms, so terms which don't step are trivially normal. This rules out looping and diverging terms, since they never stop stepping.

$\boxed{v \in \mathsf{sn}}$ $\boxed{m \in \mathsf{sn}}$ *strongly normalizing terms*

$$\textsc{sn-Val}\ \frac{\forall w.\ v \rightsquigarrow w \Rightarrow w \in \mathsf{sn}}{v \in \mathsf{sn}} \qquad \textsc{sn-Com}\ \frac{\forall n.\ m \rightsquigarrow n \Rightarrow n \in \mathsf{sn}}{m \in \mathsf{sn}}$$

$\boxed{m \rightsquigarrow_{\mathsf{sn}} n}$ *strong head reduction*

$$\textsc{sr-Thunk}\ \frac{}{\mathsf{force}\ (\mathsf{thunk}\ m) \rightsquigarrow_{\mathsf{sn}} m} \qquad \textsc{sr-Lam}\ \frac{v \in \mathsf{sn}}{(\lambda x.\ m)\ v \rightsquigarrow_{\mathsf{sn}} m[x \mapsto v]} \qquad \textsc{sr-Ret}\ \frac{v \in \mathsf{sn}}{\mathsf{let}\ x \leftarrow (\mathsf{return}\ v)\ \mathsf{in}\ m \rightsquigarrow_{\mathsf{sn}} m[x \mapsto v]}$$

$$\textsc{sr-Inl}\ \frac{v \in \mathsf{sn} \qquad n \in \mathsf{sn}}{\mathsf{case}\ (\mathsf{inl}\ v)\ \mathsf{of}\ \{\mathsf{inl}\ y \Rightarrow m\ ;\ \mathsf{inr}\ z \Rightarrow n\} \rightsquigarrow_{\mathsf{sn}} m[y \mapsto v]} \qquad \textsc{sr-Inr}\ \frac{v \in \mathsf{sn} \qquad m \in \mathsf{sn}}{\mathsf{case}\ (\mathsf{inr}\ v)\ \mathsf{of}\ \{\mathsf{inl}\ y \Rightarrow m\ ;\ \mathsf{inr}\ z \Rightarrow n\} \rightsquigarrow_{\mathsf{sn}} n[z \mapsto v]}$$

$$\textsc{sr-App}\ \frac{m \rightsquigarrow_{\mathsf{sn}} n}{m\ v \rightsquigarrow_{\mathsf{sn}} n\ v} \qquad \textsc{sr-Let}\ \frac{m \rightsquigarrow_{\mathsf{sn}} m'}{\mathsf{let}\ x \leftarrow m\ \mathsf{in}\ n \rightsquigarrow_{\mathsf{sn}} \mathsf{let}\ x \leftarrow m'\ \mathsf{in}\ n}$$

$\boxed{m \in \mathsf{ne}}$ *neutral computations*

$$\textsc{ne-Force}\ \frac{}{\mathsf{force}\ x \in \mathsf{ne}} \qquad \textsc{ne-App}\ \frac{m \in \mathsf{ne}}{m\ v \in \mathsf{ne}} \qquad \textsc{ne-Let}\ \frac{m \in \mathsf{ne}}{\mathsf{let}\ x \leftarrow m\ \mathsf{in}\ n \in \mathsf{ne}} \qquad \textsc{ne-Case}\ \frac{}{\mathsf{case}\ x\ \mathsf{of}\ \{\mathsf{inl}\ y \Rightarrow m\ ;\ \mathsf{inr}\ z \Rightarrow n\} \in \mathsf{ne}}$$

$$\textsc{sn-App-inv1}\ \frac{m\ v \in \mathsf{sn}}{m \in \mathsf{sn}} \qquad \textsc{sn-App-inv2}\ \frac{m\ v \in \mathsf{sn}}{v \in \mathsf{sn}} \qquad \textsc{sn-Let-inv1}\ \frac{\mathsf{let}\ x \leftarrow m\ \mathsf{in}\ n \in \mathsf{sn}}{m \in \mathsf{sn}} \qquad \textsc{sn-Let-inv2}\ \frac{\mathsf{let}\ x \leftarrow m\ \mathsf{in}\ n \in \mathsf{sn}}{n \in \mathsf{sn}}$$

Proof. By induction on the premises of strongly normal terms, using congruence of single-step reduction. □

Definition 4 (Neutral values)**.** A value is neutral, written $v \in \mathsf{ne}$, if it is a variable.

Lemma 9 (Preservation)**.**

1. *If* $v \rightsquigarrow w$ *and* $v \in \mathsf{ne}$ *then* $w \in \mathsf{ne}$.
2. *If* $m \rightsquigarrow n$ *and* $m \in \mathsf{ne}$ *then* $n \in \mathsf{ne}$.

Proof. By mutual induction on the single-step reductions. □

Definition 5 (Strongly neutral computations)**.** A strongly neutral computation, written $m \in \mathsf{sne}$, is a computation that is both neutral and strongly normalizing, *i.e.* $m \in \mathsf{ne}$ and $m \in \mathsf{sn}$.

$\boxed{v \in \mathsf{sn}}$ *admissible strongly normal values*

$$\textsc{sn-Var}\ \frac{}{x \in \mathsf{sn}} \qquad \textsc{sn-Unit}\ \frac{}{\mathsf{unit} \in \mathsf{sn}} \qquad \textsc{sn-Inl}\ \frac{v \in \mathsf{sn}}{\mathsf{inl}\ v \in \mathsf{sn}} \qquad \textsc{sn-Inr}\ \frac{v \in \mathsf{sn}}{\mathsf{inr}\ v \in \mathsf{sn}} \qquad \textsc{sn-Thunk}\ \frac{m \in \mathsf{sn}}{\mathsf{thunk}\ m \in \mathsf{sn}}$$

$\boxed{m \in \mathsf{sn}}$ *admissible strongly normal computations*

$$\textsc{sn-Lam}\ \frac{m \in \mathsf{sn}}{\lambda x.\, m \in \mathsf{sn}} \qquad \textsc{sn-Ret}\ \frac{v \in \mathsf{sn}}{\mathsf{return}\ v \in \mathsf{sn}} \qquad \textsc{sn-Force}\ \frac{v \in \mathsf{sn}}{\mathsf{force}\ v \in \mathsf{sn}} \qquad \textsc{sn-App}\ \frac{m \in \mathsf{sne} \qquad v \in \mathsf{sn}}{m\ v \in \mathsf{sn}} \qquad \textsc{sn-Let}\ \frac{m \in \mathsf{sne} \qquad n \in \mathsf{sn}}{\mathsf{let}\ x \leftarrow m\ \mathsf{in}\ n \in \mathsf{sn}}$$

$$\textsc{sn-Case}\ \frac{v \in \mathsf{sne} \qquad m \in \mathsf{sn} \qquad n \in \mathsf{sn}}{\mathsf{case}\ v\ \mathsf{of}\ \{\mathsf{inl}\ y \Rightarrow m\ ;\ \mathsf{inr}\ z \Rightarrow n\} \in \mathsf{sn}}$$

Proof. Rules SN-App, SN-Let, and SN-Case are proven by double induction on the two derivations of strongly normal terms. Intuitively, we want to show that if the conclusion steps, then it steps to a strongly normal term, knowing by the induction hypotheses that if their subterms reduce, then they also reduce to strongly normal terms. Neutrality of the term in head position eliminates cases where the conclusion β-reduces, leaving only the congruent reductions. Because single-step reduction only steps in one subterm, we only need the induction hypothesis for that reducing subterm, so the double induction is really a lexicographic induction on the two derivations. We additionally require Preservation to carry along neutrality when the heads reduce in cases SN-App and SN-Let. All other cases are direct by construction or by induction on their sole premise. □

Lemma 10 (Antisubstitution (sn))**.** *If* $m[x \mapsto v] \in \mathsf{sn}$ *and* $v \in \mathsf{sn}$ *then* $m \in \mathsf{sn}$.

Proof. By induction on the derivation of $m[x \mapsto v] \in \mathsf{sn}$. □

$\boxed{m \in \mathsf{sn}}$ *head expansion*

$$\textsc{sn-Force-Thunk}\ \frac{m \in \mathsf{sn}}{\mathsf{force}\ (\mathsf{thunk}\ m) \in \mathsf{sn}} \qquad \textsc{sn-App-Lam}\ \frac{v \in \mathsf{sn} \qquad m[x \mapsto v] \in \mathsf{sn}}{(\lambda x.\, m)\ v \in \mathsf{sn}} \qquad \textsc{sn-Let-Ret}\ \frac{v \in \mathsf{sn} \qquad m[x \mapsto v] \in \mathsf{sn}}{\mathsf{let}\ x \leftarrow (\mathsf{return}\ v)\ \mathsf{in}\ m \in \mathsf{sn}}$$

$$\textsc{sn-Case-Inl}\ \frac{v \in \mathsf{sn} \qquad m[y \mapsto v] \in \mathsf{sn} \qquad n \in \mathsf{sn}}{\mathsf{case}\ (\mathsf{inl}\ v)\ \mathsf{of}\ \{\mathsf{inl}\ y \Rightarrow m\ ;\ \mathsf{inr}\ z \Rightarrow n\} \in \mathsf{sn}} \qquad \textsc{sn-Case-Inr}\ \frac{v \in \mathsf{sn} \qquad m \in \mathsf{sn} \qquad n[z \mapsto v] \in \mathsf{sn}}{\mathsf{case}\ (\mathsf{inr}\ v)\ \mathsf{of}\ \{\mathsf{inl}\ y \Rightarrow m\ ;\ \mathsf{inr}\ z \Rightarrow n\} \in \mathsf{sn}}$$

Proof. Rule SN-FORCE-THUNK holds directly by induction on the premise; the remaining proofs are more complex. First, given the premise $m[x \mapsto v] \in \mathsf{sn}$ (or $n[z \mapsto v]$), use Antisubstitution (sn) to obtain $m \in \mathsf{sn}$. Then proceed by double (or triple) induction on the derivations of $v \in \mathsf{sn}$ and $m \in \mathsf{sn}$ (and $n \in \mathsf{sn}$ for the case rules). Similarly to the admissible strongly normal rules, these are lexicographic inductions, except $m[x \mapsto v] \in \mathsf{sn}$ (or $n[z \mapsto v]$) is used to satisfy the β-reduction cases. □

Lemma 11 (Confluence)**.** *If $m \rightsquigarrow n_1$ and $m \rightsquigarrow_{\mathsf{sn}} n_2$, then either $n_1 = n_2$, or there exists some m' such that $n_1 \rightsquigarrow_{\mathsf{sn}} m'$ and $n_2 \rightsquigarrow m'$.*

Proof. By induction on the derivation of $m \rightsquigarrow_{\mathsf{sn}} n_2$, then by cases on the derivation of $m \rightsquigarrow n_1$. □

$\boxed{m \in \mathsf{sn}}$ *backward closure in head position*

$$\textsc{sn-App-bwd}\ \frac{m \rightsquigarrow_{\mathsf{sn}} n \qquad v \in \mathsf{sn} \qquad n\ v \in \mathsf{sn}}{m\ v \in \mathsf{sn}} \qquad \textsc{sn-Let-bwd}\ \frac{m \rightsquigarrow_{\mathsf{sn}} m' \qquad n \in \mathsf{sn} \qquad \mathsf{let}\ x \leftarrow m'\ \mathsf{in}\ n \in \mathsf{sn}}{\mathsf{let}\ x \leftarrow m\ \mathsf{in}\ n \in \mathsf{sn}}$$

Proof. First, use rules SN-APP-INV and SN-LET-INV to obtain $m \in \mathsf{sn}$. Then proceed by double induction on the derivations of $m \in \mathsf{sn}$ and $v \in \mathsf{sn}/n \in \mathsf{sn}$, again as lexicographic induction. We want to show that if the conclusion steps, then it steps to a strongly normal term. Strong reduction of the head position eliminates the cases of β-reduction, leaving the cases where the head position steps or the other position steps. If the head position steps, we use Confluence to join the strong reduction and the single-step reduction together, then use the first induction hypothesis. Otherwise, we use the second induction hypothesis. We need the last premise to step through either of the subterms, since we have no congruence rule for when the head position is not neutral. □

Lemma 12 (Backward closure)**.** *If $m \rightsquigarrow_{\mathsf{sn}} n$ and $n \in \mathsf{sn}$ then $m \in \mathsf{sn}$.*

Proof. By induction on the derivation of $m \rightsquigarrow_{\mathsf{sn}} n$. The cases correspond exactly to each of rules SN-FORCE-THUNK, SN-APP-LAM, SN-LET-RET, SN-CASE-INL, SN-CASE-INR, SN-APP-BWD, and SN-LET-BWD. □

Lemma 13 (Soundness (SNe))**.** *If $m \in \mathsf{SNe}$ then $m \in \mathsf{ne}$.*

Proof. By induction on the derivation of $m \in \mathsf{SNe}$. □

Theorem 14 (Soundness (SN))**.**

1. *If $m \in \mathsf{SNe}$ then $m \in \mathsf{sn}$.*
2. *If $v \in \mathsf{SN}$ then $v \in \mathsf{sn}$.*
3. *If $m \in \mathsf{SN}$ then $m \in \mathsf{sn}$.*
4. *If $m \rightsquigarrow n$ then $m \rightsquigarrow_{\mathsf{sn}} n$.*

Proof. By mutual induction on the derivations of $m \in \mathsf{SNe}$, $v \in \mathsf{SN}$, $m \in \mathsf{SN}$, and $m \rightsquigarrow n$. The cases for the first three correspond to the admissible strongly normal rules, using Soundness (SNe) as needed, except for the SN-RED case, which uses Backward closure. The cases for strong reduction hold by construction. □

Corollary 15. *If $\Gamma \vdash v : A$ then $v \in \mathsf{sn}$, and if $\Gamma \vdash m : B$ then $m \in \mathsf{sn}$.*

7 Discussion