Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 351 (2020) 187–203

www.elsevier.com/locate/entcs

Strong Normalization for the Simply-Typed Lambda Calculus in Constructive Type Theory Using Agda

Sebastián Urciuoli[1] Álvaro Tasistro Nora Szasz[2]

Universidad ORT Uruguay
Montevideo, Uruguay

Abstract

We consider a pre-existing formalization in Constructive Type Theory of the pure Lambda Calculus in its presentation in first order syntax with only one sort of names and alpha-conversion based upon multiple substitution, as well as of the system of assignment of simple types to terms. On top of it, we formalize a slick proof of strong normalization given by Joachimski and Matthes whose main lemma proceeds by complete induction on types and subordinate induction on a characterization of the strongly normalizing terms which is in turn proven sound with respect to their direct definition as the accessible part of the relation of one-step beta reduction. The proof of strong normalization itself is thereby allowed to consist just of a direct induction on the type system. The whole development has been machine-checked using the system Agda. We assess merits and drawbacks of the approach.

Keywords: Formal Metatheory, Lambda Calculus, Constructive Type Theory

1 Introduction

In [3] we have presented a formalization of the Lambda Calculus in Constructive Type Theory using first-order syntax, with only one sort of names for both bound and free variables, without identifying terms up to alpha conversion, and using multiple (simultaneous) substitution as fundamental operation.

It was then our aim to investigate whether this very concrete approach was in any way amenable to full formalization. The approach is historically rooted in the detailed meta-theoretical study of the calculus by Curry and Feys [7] where (unary) substitution was given a well-known definition as a total meta-operation on terms on top of which all the relevant relations of the calculus where defined, starting with alpha conversion. That definition of substitution proceeds by recursion on the

[1] Partially supported by Agencia Nacional de Investigación e Innovación (ANII), Uruguay.

[2] Email: {urciuoli,tasistro,szasz}@ort.edu.uy

https://doi.org/10.1016/j.entcs.2020.08.010

size of terms, but its full justification requires a simultaneous proof that its use as a renaming operation (i.e. when the substituted term is a variable) does not affect the size of the term wherein it is effected. This makes its formalization in e.g. Constructive Type Theory and using any of the proof assistants by now available, extremely difficult or even impossible.

However, the use of simultaneous multiple substitution as suggested in e.g. [11] brought about the possibility of achieving a development of the meta-theory of the calculus that was at the same time fully formal –to the extent of being close to a machine-checkable version– and humanly readable. Indeed, as explained in [11,3] the use of this kind of substitution makes it possible to avoid the recursion on the size of terms since, when crossing over lambdas the bound name is changed to an appropriate one and the corresponding renaming recorded into an enlarged, multiple substitution that goes on to operate on the body of the abstraction. Hence the effect of substitution can be given a simple definition by structural recursion which makes it plausible that the meta-theory of the calculus can be pursued using also simple forms of induction. The fact that bound names are always changed permits to avoid case analyses too.

Our work in [3,6] verified the preceding hypotheses to some extent: we were able to carry out full formalizations of the meta-theory of the pure Lambda Calculus up to the Standardization Theorem, and of the simply typed Lambda Calculus (more precisely, of the system of assignment of simple types to terms) up to the Subject-Reduction Theorem. Our main conclusion from those efforts has been that, in spite of having to explicitly consider alpha conversion, the workload was manageable, the formal machine-checked proofs not extremely lengthy, and their presentation in detailed mathematical English readable and reasonably complete. We also generated a basic Agda library [2] that was successfully employed and enlarged by a Master's student in [6].

In the present paper we wish to continue the preceding line of work, with the purposes of:

- testing the foregoing hypotheses against challenges of increasing complexity, and
- making progress towards a comprehensive corpus of fully formal meta-theory of the Lambda Calculus.

We therefore tackle the problem of formally proving Strong Normalization for the system of assignment of simple types to terms. The method of proof we choose to formalize is due to [8] where the authors present slick proofs of weak and strong normalization which proceed by simple induction on the type system. These proofs use a main lemma that allows to deal with the difficult case of applications in the induction, via a separate complete induction on types, with a subordinate induction which, in the case of strong normalization, proceeds on a characterization of the strong normalizing terms originally given in [13].

In [8], vector notation is used for presenting terms which we shall avoid for a matter of convenience of the formalization. Also, –like in most of meta-theory developments– terms are identified under alpha conversion, with the usual conven-

tions for choice of bound names being made use of without further ado. Then, for example, variable capture is not take into account –which of course, we will have to. We wish to test whether:

- our formalization can still be kept within limits of reasonable complexity,
- our pre-existing Agda library makes the grade as to continue supporting this additional development, and
- the effort is within the reach of a newcomer with limited experience in full formalization, the use of proof assistants, and Agda in particular. Actually, the work forms part of the Master's thesis of the first author. The whole development has been formalised in Constructive Type Theory and machine-checked in the system Agda [10]. The corresponding code is available at `https://github.com/surciuoli/lambda-metatheory`.

We proceed now to the development itself. Its structure is as follows: in the next section we present the basic definitions, and results that are used in this formalization –some from [3], and some new developments. In Section 3 we give two definitions of the strongly normalizing terms. The first is the more natural one, defined as the accessible part of the one-step beta reduction, and the other is a syntactic characterization originally given (in vector notation) in [13]. Our formulation of the latter is mutual with the predicate characterizing strongly normalizing neutral terms and the strong head reduction. We formalize the proof of soundness of the second definition with respect to the original one in Section 4. In Section 5 we make use of the syntactic characterization to obtain a proof of the Strong Normalization Theorem for typable terms in the system of assignments of simple types, which ultimately consists of a direct induction on the type system. Finally, in Section 6 we present the overall conclusions.

2 Preliminaries

2.1 Syntax

We start with a denumerable type V of names, also to be called variables; i.e. for concreteness we can put $\mathsf{V} =_{def} \mathbb{N}$ or $\mathsf{V} =_{def} \mathsf{String}$. Letters x, y, z with primes or subindices shall stand for variables. Terms are defined inductively as usual —below we show a grammar for the abstract syntax:

Definition 2.1 *The terms in Λ are defined by the following grammar:*

$$M, N ::= x \mid MN \mid \lambda x M.$$

M, N, P, $Q\ldots$ range over lambda terms. In the concrete syntax we assume the usual convention according to which application binds tighter than abstraction.

Definition 2.2 *That a variable x is free in M is written $x * M$ and its opposite is written $x\#M$, to be pronounced x fresh in M as in nominal techniques. Both relations are defined inductively in a standard manner.*

2.2 Substitutions

We work with multiple (simultaneous) substitutions, i.e. mappings associating a term to every variable. Actually, the substitutions we must handle are identity almost everywhere, and so we can generate them by starting up from the empty substitution ι, which maps each variable to itself as a term, and applying the update operation defined as follows:

Definition 2.3 (Substitution Update) *$(\sigma, x{:=}M)$ is the substitution defined by:*

$$\begin{aligned}(\sigma, x{:=}M)\ x &= M\\ (\sigma, y{:=}M)\ x &= \sigma x,\ \text{ if } x \neq y.\end{aligned}$$

Whenever needed, we will write $[x{:=}N]$ as syntactic sugar for $(\iota, x{:=}N)$.
Also, we often consider the *restriction* of a substitution σ to the free variables of a given term M, which is just written (σ, M). Then, we can extend fresh and free variables to restrictions as well:

Definition 2.4 *A variable x is fresh in (σ, M) and written $x\#(\sigma, M)$ iff for all $y * M$, then $x\#\sigma y$. On the contrary, x is said to be free in (σ, M) and written $x * (\sigma, M)$ iff there exists some $y * M$ such that $x * \sigma y$.*

Definition 2.5 (Effect of Substitution on Terms) *The effect of a substitution σ on a term M is defined by recursion on M:*

$$\begin{aligned}x \cdot \sigma &= \sigma x\\ (MN) \cdot \sigma &= (M \cdot \sigma)(N \cdot \sigma)\\ (\lambda x M) \cdot \sigma &= \lambda z (M \cdot (\sigma, x{:=}z)) \quad \text{with } z = \chi(\sigma, \lambda x M)\end{aligned} \tag{1}$$

In equation (1), notice that the bound variable x is always replaced with a new one. The new variable z is obtained by means of a choice function χ that computes the first variable fresh in $(\sigma, \lambda x M)$, so that it does not capture any of the names introduced into its scope by effect of the substitution (see [3] for more details).
Besides implementing capture-avoidance, the use of multiple substitutions and uniform renaming of bound variables allows us to employ simple structural induction, avoiding case analyses when reasoning with substitutions. We will use $M\sigma$ as syntactic sugar for $M \cdot \sigma$ and, in concrete syntax, substitution will bind tighter than application. The following properties about substitutions are proven in [3]:

Lemma 2.6

1. $\chi(\sigma, M)\#(\sigma, M)$
2. $\chi(\iota, M)\#M$
3. $M(\sigma, x{:=}N\sigma) = M[x{:=}N]\sigma$

Definition 2.7 (Equality on Restrictions) $(\sigma, M) = (\sigma', M')$ *iff M and M' share the same free variables and $(\forall x * M)\ \sigma x = \sigma' x$.*

Then, $\sigma = \sigma' \downarrow M$ will be sugar for $(\sigma, M) = (\sigma', M)$.

2.3 Alpha Conversion

Definition 2.8 (Alpha-conversion)

$$\sim\mathtt{v}:\ \frac{}{x \sim_\alpha x} \qquad \sim\mathtt{app}:\ \frac{M \sim_\alpha M' \qquad N \sim_\alpha N'}{MN \sim_\alpha M'N'} \qquad \sim\lambda:\ \frac{M[x{:=}z] \sim_\alpha N[y{:=}z]}{\lambda xM \sim_\alpha \lambda yN}\ (*)$$

(*) with $z\#\lambda xM, \lambda yN$.

Alpha conversion is extended to restrictions by the following definition:

Definition 2.9 $\sigma \sim_\alpha \sigma' \downarrow M = (\forall x * M)\ \sigma x \sim_\alpha \sigma' x$

Then, the following lemmas –also proven in [3]– hold:

Lemma 2.10

1. $\iota \sim_\alpha \iota \downarrow M$
2. $\sigma \sim_\alpha \sigma' \downarrow M \wedge N \sim_\alpha P \Rightarrow (\sigma, x{:=}N) \sim_\alpha (\sigma', x{:=}P) \downarrow M$

Next we show some results about $\sim_\alpha$ proven in [3] and needed in our development:

Lemma 2.11

1. $\sim_\alpha$ *is an equivalence relation.*
2. $y\#(\sigma, \lambda xM) \Rightarrow M(\sigma, x{:=}y)[y{:=}N] \sim_\alpha M(\sigma, x{:=}N)$
3. $y\#\lambda xM \Rightarrow \lambda xM \sim_\alpha \lambda y(M[x{:=}y])$
4. $M \sim_\alpha N \Rightarrow \lambda xM \sim_\alpha \lambda xN$
5. $y\#(\sigma, \lambda xM) \Rightarrow (\lambda xM)\sigma \sim_\alpha \lambda y(M(\sigma, x{:=}y))$
6. $x\#M \wedge M \sim_\alpha N \Rightarrow x\#N$
7. $M \sim_\alpha M' \wedge \sigma \sim_\alpha \sigma' \downarrow M \Rightarrow M\sigma \sim_\alpha M'\sigma'$
8. $M \sim_\alpha N \Rightarrow M\sigma = N\sigma$
9. $M \sim_\alpha M\iota$

From these, some useful corollaries are immediate:

Corollary 2.12

1. $z\#(\sigma, \lambda yM) \Rightarrow M(\sigma, y{:=}z)[z{:=}N\sigma] \sim_\alpha M[y{:=}N]\sigma$
2. $(\lambda xM)N \sim_\alpha (\lambda yM')N' \Rightarrow M[x{:=}N] \sim_\alpha M'[y{:=}N']$
3. $\lambda xM \sim_\alpha \lambda yN \Rightarrow M[x{:=}y] \sim_\alpha N$

2.4 Beta Reduction

(One step) β-reduction is denoted $\rightarrow_\beta$, and it is defined by the following rules:

Definition 2.13 (Beta Reduction)

$$\beta:\ \frac{}{(\lambda xM)N \rightarrow_\beta M[x{:=}N]} \qquad \mathtt{appL}:\ \frac{M \rightarrow_\beta M'}{MN \rightarrow_\beta M'N} \qquad \mathtt{appR}:\ \frac{N \rightarrow_\beta N'}{MN \rightarrow_\beta MN'}$$

$$\lambda\colon \frac{M \to_\beta M'}{\lambda x M \to_\beta \lambda x M'}$$

Multi-step reduction is written $\twoheadrightarrow$ and defined below:

Definition 2.14 (Multi-step Reduction)

$$\frac{M \sim_\alpha N}{M \twoheadrightarrow N} \qquad \frac{M \to_\beta N}{M \twoheadrightarrow N} \qquad \frac{M \twoheadrightarrow N \qquad N \twoheadrightarrow P}{M \twoheadrightarrow P}$$

We extend $\twoheadrightarrow$ to substitutions as follows:

Definition 2.15 (Reductions on Substitutions) $\sigma \twoheadrightarrow \sigma' \;=\; (\forall x)\ \sigma x \twoheadrightarrow \sigma' x.$

The following results from [3] involving $\to_\beta$ will be needed in our development:

Lemma 2.16

1. $x\#M \wedge M \to_\beta N \Rightarrow x\#N$
2. $x\#(\sigma, M) \wedge M \to_\beta N \Rightarrow x\#(\sigma, N)$
3. $\sigma \twoheadrightarrow \sigma' \Rightarrow M\sigma \twoheadrightarrow M\sigma'$
4. $M \twoheadrightarrow M' \Rightarrow MN \twoheadrightarrow M'N$

From now on, except when explicitly stated, all results presented constitute new developments.

Lemma 2.17 (Compatibility of substitution with $\to_\beta$, up to $\sim_\alpha$)

$$M \to_\beta N \Rightarrow (\exists P)\ M\sigma \to_\beta P \sim_\alpha N\sigma$$

Proof. By induction on the derivation of $M \to_\beta N$.

- Case β: $(\lambda x M)N \to_\beta M[x{:=}N]$. Let $P = M(\sigma,\ x{:=}z)(\iota,\ z{:=}N\sigma)$ with $z = \chi(\sigma, \lambda x M)$. Then, we need to show: (i) $((\lambda x M)N)\sigma \to_\beta P$ and (ii) $P \sim_\alpha M[x{:=}N]\sigma$. (i) By definition of the effect of substitution (Def. 2.5) we have $((\lambda x M)N)\sigma = (\lambda z(M(\sigma,\ x{:=}z))N\sigma$ and by β rule, $(\lambda z(M(\sigma,\ x{:=}z))N\sigma \to_\beta M(\sigma,\ x{:=}z)(\iota,\ z{:=}N\sigma)$. (ii) By Corollary 2.12.1.
- Case `appL`: $MN \to_\beta M'N$ follows from $M \to_\beta M'$. By ind. hyp. we know there exists some Q s.t. $M\sigma \to_\beta Q \sim_\alpha M'\sigma$. Using `appL` rule we get $(MN)\sigma \to_\beta QN\sigma$ and by $\sim$`app` it follows $QN\sigma \sim_\alpha (M'N)\sigma$.
- Case `appR` : $MN \to_\beta MN'$ follows from $N \to_\beta N'$. Analogous to the `appL` case.
- Case λ: $\lambda x M \to_\beta \lambda x N$ follows from $M \to_\beta N$. We have to show: (i) $(\lambda x M)\sigma \to_\beta P$ and (ii) $P \sim_\alpha (\lambda x N)\sigma$, for some P. First, by ind. hyp. we know that there exists some P' s.t. $M(\sigma,\ x{:=}z) \to_\beta P' \sim_\alpha N(\sigma,\ x{:=}z)$. Let $P = \lambda z P'$. On the one side, (i) follows immediately by λ rule and Definition 2.5. (ii) On the other side, by Lemma 2.11.4 we have $\lambda z P' \sim_\alpha \lambda z N(\sigma,\ x{:=}z)$. In addition, by Lemma 2.6.1 we get $z\#(\sigma, \lambda x M)$, thus by Lemma 2.16.2 we obtain $z\#(\sigma, \lambda x N)$. Finally, by Lemma 2.11.5 we have $\lambda z P' \sim_\alpha (\lambda x N)\sigma$, as desired. □

Lemma 2.18 (Commutativity of $\sim_\alpha$ and $\to_\beta$)

$$M \sim_\alpha N \to_\beta P \Rightarrow (\exists Q)\ M \to_\beta Q \sim_\alpha P$$

Proof. By induction on M. We show only the proofs of the interesting cases.

- Case $\lambda x M \sim_\alpha \lambda y N \to_\beta \lambda y P$. On the one hand, $\lambda y N \to_\beta \lambda y P$ must follow from $N \to_\beta P$ (Def. 2.13) and by Lemma 2.17 there is a term Q s.t. $N[y{:=}x] \to_\beta Q \sim_\alpha P[y{:=}x]$. In addition, by Corollary 2.12.3 we get $M \sim_\alpha N[y{:=}x]$. Then, by ind. hyp. using $M \sim_\alpha N[y{:=}x] \to_\beta Q$ we obtain $M \to_\beta Q' \sim_\alpha Q$, for some Q', and by Definition 2.13 we get $\lambda x M \to_\beta \lambda x Q'$. On the other hand, by hypothesis, using Lemmas 2.11.6 and 2.16.1 we get $x \# \lambda y P$, and by Lemma 2.11.3, $\lambda y P \sim_\alpha \lambda x P[y{:=}x]$. Finally, $\lambda x Q' \sim_\alpha \lambda y P$ holds by Lemma 2.11.4 and transitivity of $\sim_\alpha$.
- Case $(\lambda x M) M' \sim_\alpha (\lambda y N) N' \to_\beta N[y{:=}N']$. Let $Q = M[x{:=}M']$. By Def. 2.13 we get $(\lambda x M) M' \to_\beta Q$ and using Lemma 2.12.2 we have $Q \sim_\alpha N[y{:=}N']$. □

Lemma 2.19 (Commutativity of $\sim_\alpha$ and $\twoheadrightarrow$)

$$M \sim_\alpha N \twoheadrightarrow P \Rightarrow (\exists Q)\ M \twoheadrightarrow Q \sim_\alpha P$$

Proof. Follows easily by cases on $N \twoheadrightarrow P$, using Lemma 2.18. □

2.5 Unary Substitutions

It turns out convenient to introduce substitutions that arise in the process of reducing a β-redex. They will consist of a pair associating a variable to a term accompanied by several variable renamings.

Definition 2.20 *A substitution σ is unary of variable x and term M –written as* $\mathsf{Unary}\,\sigma\,x\,M$*– iff $\sigma x = M$ and, for all $y \neq x$, σy is a variable.*

Lemma 2.21 (Unary Substitution Lemmas)

1. $\mathsf{Unary}\,[x{:=}M]\,x\,M$
2. $\mathsf{Unary}\,\sigma\,x\,M \Rightarrow \mathsf{Unary}\,(\sigma, x{:=}N)\,x\,N$
3. $y \neq x \wedge \mathsf{Unary}\,\sigma\,x\,M \Rightarrow \mathsf{Unary}\,(\sigma, y{:=}z)\,x\,M$
4. $\mathsf{Unary}\,\sigma\,x\,y \Rightarrow \mathsf{Unary}\,(\sigma, z{:=}M)\,z\,M$

Proof. Trivially by definition. □

3 Inductive Definitions of Strongly Normalizing Terms

Our goal is to prove that all terms typable in the system of assignment of simple types are strongly normalizing. In order to do that, we will proceed as follows:
In this section we present two definitions of the strongly normalizing terms. The first one (predicate sn) is naturally introduced as the accessible part of the one-step beta reduction. The second one (predicate SN) is an alternative syntactic characterization introduced in [13], originally given in vector notation and adapted in [1] in a somewhat different context (i.e. using typed reductions).

We shall presently give the two definitions together with some of their properties, in order to eventually prove the soundness of the second definition with respect to the original one, i.e. that for all terms M, $\mathsf{SN}\,M$ implies $\mathsf{sn}\,M$, in the next section. Later, in Section 5, we will complete the Strong Normalization proof by showing that all terms typable in the system of simple types assignment satisfy the second definition (SN).

3.1 Strongly Normalizing terms

A term is strongly normalizing if and only if it is in the accessible part of $\to_\beta$:

Definition 3.1 (Accessibility definition of strongly normalizing terms)

$$\frac{(\forall N)\ (M \to_\beta N \Rightarrow \mathsf{sn}\,N)}{\mathsf{sn}\,M}$$

Lemma 3.2 (Closure of sn under $\sim_\alpha$) $\mathsf{sn}\,M \wedge M \sim_\alpha N \Rightarrow \mathsf{sn}\,N$.

Proof. By induction on $\mathsf{sn}\,M$. Take N such that $M \sim_\alpha N$ and P such that $N \to_\beta P$. By Lemma 2.18 we have Q s.t. $M \to_\beta Q \sim_\alpha P$. Hence $\mathsf{sn}\,Q$ and by ind. hyp., since $Q \sim_\alpha P$, $\mathsf{sn}\,P$ as desired. □

Lemma 3.3 (Properties of sn)

1. $\mathsf{sn}\,x$
2. $\mathsf{sn}\,M \ \wedge\ M \twoheadrightarrow N \Rightarrow \mathsf{sn}\,N$
3. $\mathsf{sn}\,(MN) \Rightarrow \mathsf{sn}\,M \wedge \mathsf{sn}\,N$
4. $\mathsf{sn}\,M \Rightarrow \mathsf{sn}\,(\lambda x M)$

Proof. (3.3.1) By vacuity, since there is no possible way of having $x \to_\beta M$ for any x. (3.3.2) By induction on $M \twoheadrightarrow N$. (3.3.3) By induction on $\mathsf{sn}\,(MN)$. (3.3.4) By induction on $\mathsf{sn}\,M$. □

3.2 Strongly Normalizing terms: a syntactic definition

Now we introduce the second definition of strongly normalizing terms. In order to do this, we must define three mutually inductive predicates: strongly normalizing terms (SN), strongly normalizing neutral terms with head x (SNe_x) and the relation of strong head reduction ($\to_{\mathsf{SN}}$).

Definition 3.4 (SN, SNe and $\to_{\mathsf{SN}}$)

$$\mathtt{v}\colon \frac{}{\mathsf{SNe}_x\,x} \qquad \mathtt{app}\colon \frac{\mathsf{SNe}_x\,M \qquad \mathsf{SN}\,N}{\mathsf{SNe}_x\,MN}$$

$$\mathtt{ne}\colon \frac{\mathsf{SNe}_x\,M}{\mathsf{SN}\,M} \qquad \lambda\colon \frac{\mathsf{SN}\,M}{\mathsf{SN}\,\lambda x M} \qquad \mathtt{exp}\colon \frac{M \to_{\mathsf{SN}} N \qquad \mathsf{SN}\,N}{\mathsf{SN}\,M}$$

$$\beta\colon \frac{\mathsf{SN}\,N}{(\lambda x M)N \to_{\mathsf{SN}} M[x{:=}N]} \qquad \mathtt{appL}\colon \frac{M \to_{\mathsf{SN}} M'}{MN \to_{\mathsf{SN}} M'N}$$

$$\alpha\colon \frac{M \to_{\mathsf{SN}} R \qquad R \sim_\alpha N}{M \to_{\mathsf{SN}} N}$$

4 Soundness of SN

In this section we will show that SN is sound with respect to sn, i.e, every term in the first predicate is also in the second one. In order to achieve this, we first need to introduce two notions, namely that of neutral term and that of strongly normalizing head reductions that do not step inside abstractions.

4.1 Neutral terms

Let us call *neutral* the iterated applications having a variable as head:

Definition 4.1 (Neutral terms)

$$\frac{}{\mathsf{ne}\, x\, x} \qquad \frac{\mathsf{ne}\, x\, M}{\mathsf{ne}\, x\, MN}$$

We will omit the head and just write $\mathsf{ne}\, M$ whenever it results convenient.
Neutral terms enjoy the following properties:

Lemma 4.2 (Closure of ne under $\to_\beta$) $\mathsf{ne}\, M \ \wedge\ M \to_\beta N \Rightarrow \mathsf{ne}\, N$

Proof. By induction on $\mathsf{ne}\, M$. □

Lemma 4.3 (Closure of sn/ne under app.) $\mathsf{ne}\, M \wedge \mathsf{sn}\, M \wedge \mathsf{sn}\, N \Rightarrow \mathsf{sn}\, MN$.
Proof. By lexicographic induction on $\mathsf{sn}\, M$, $\mathsf{sn}\, N$. Assume $MN \to_\beta P$ and proceed by cases to prove $\mathsf{sn}\, P$:

- In case $P = M'N$ with $M \to_\beta M'$, using Lemma 4.2 we obtain $\mathsf{ne}\, M'$. By definition of $\mathsf{sn}\, M$ we get $\mathsf{sn}\, M'$ and then, by ind. hyp., $\mathsf{sn}\, P$.
- In case $P = MN'$ with $N \to_\beta N'$ proceed analogously using $\mathsf{sn}\, N'$. □

4.2 Strongly Normalizing head reductions

Definition 4.4 (Strongly Normalizing head reductions) *$\to_{\mathsf{sn}}$ is the reduction relation defined as:*

$$\beta\colon \frac{\mathsf{sn}\, N}{(\lambda x M)N \to_{\mathsf{sn}} M[x{:=}N]} \qquad \mathtt{appL}\colon \frac{M \to_{\mathsf{sn}} M'}{MN \to_{\mathsf{sn}} M'N} \qquad \alpha\colon \frac{M \to_{\mathsf{sn}} R \qquad R \sim_\alpha N}{M \to_{\mathsf{sn}} N}$$

Lemma 4.5 (Confluence) *Let $M \to_{\mathsf{sn}} N$ and $M \to_\beta P$. Then either $N \sim_\alpha P$ or there exists Q s.t. $P \to_{\mathsf{sn}} Q$ and $N \twoheadrightarrow Q$.*

In other words, let Δ_L be the outer-leftmost redex of M not inside of an abstraction: then N is obtained from M by contracting Δ_L. Let Δ be the redex contracted in $M \to_\beta P$. If $\Delta = \Delta_L$, then $N = P$. Otherwise, let Q be obtained by contracting Δ_L from P, i.e. $P \to_{\mathsf{sn}} Q$. If Δ was discarded in $M \to_{\mathsf{sn}} N$, then $N = Q$. If not, then necessarily $N \to_\beta Q$ is derived by contracting Δ –and possibly other redexes in it.

Proof. By induction on $M \to_{\mathsf{sn}} N$ and subordinate analysis of $M \to_\beta P$.

- Case β: Suppose $\mathsf{sn}\, N$.
  - If $(\lambda x M)N \to_\beta P$ is obtained by rule β then we get $P = M[x{:=}N]$ immediately.

- If $(\lambda x M)N \to_\beta P$ is obtained by rule appL then P is $(\lambda x M')N$ and $M \to_\beta M'$. Let $Q = M'[x{:=}N]$. By Lemma 2.17 and Definition 2.14 it follows that $M[x{:=}N] \twoheadrightarrow Q$ and then, using rule β of Definition 4.4, we get $P \to_{\mathsf{sn}} Q$.
- If $(\lambda x M)N \to_\beta P$ is obtained by rule appR then P is $(\lambda x M)N'$ and $N \to_\beta N'$. Let $Q = M[x{:=}N']$. Now, on the one hand, by Lemma 2.17 we have $M[x{:=}N] \twoheadrightarrow Q$. And, on the other, since $\mathsf{sn}\, N'$ follows from $\mathsf{sn}\, N$ and Lemma 3.3.2, we can use again rule β of Definition 4.4 to get $P \to_{\mathsf{sn}} Q$.

- Case appL: Suppose $M \to_{\mathsf{sn}} M'$.
  - $MN \to_\beta P$ cannot be obtained using rule β, for in such case M would be an abstraction. But we have $M \to_{\mathsf{sn}} M'$ and $\to_{\mathsf{sn}}$ does not proceed on abstractions.
  - If $MN \to_\beta P$ follows from rule appL then P is $M''N$ and we have $M \to_\beta M''$. The ind. hyp. gives us two possibilities: either $M' \sim_\alpha M''$ or there exists some Q' such that $M' \twoheadrightarrow Q'$ and $M'' \to_{\mathsf{sn}} Q'$. In the first case, it follows immediately $M'N \sim_\alpha M''N$. Otherwise, let $Q = Q'N$, then by Definitions 4.4 and 2.14 we get $M'N \twoheadrightarrow Q$ and $M''N \to_{\mathsf{sn}} Q$.
  - If $MN \to_\beta P$ is obtained by rule appR then P is MN' and we have $N \to_\beta N'$. Choose then $Q = M'N'$ and use definitions of $\twoheadrightarrow$ and $\to_{\mathsf{sn}}$.
- Case α: Suppose $M \to_{\mathsf{sn}} R$ and $R \sim_\alpha N$. Suppose further $M \to_\beta P$. The ind. hyp. gives either $R \sim_\alpha P$ or there exists Q s.t. $R \twoheadrightarrow Q$ and $P \to_{\mathsf{sn}} Q$. In the first case, from $R \sim_\alpha N$ and $R \sim_\alpha P$ we obtain $N \sim_\alpha P$. In the other case, since $R \sim_\alpha N$, from $R \twoheadrightarrow Q$ it follows also $N \twoheadrightarrow Q$ by Lemma 2.19. □

4.3 The Proof of Soundness

In order to prove that SN implies sn, we also need corresponding proofs of soundness of SNe and $\to_{\mathsf{SN}}$ with respect to sn and $\to_{\mathsf{sn}}$. The difficulty lies in the case corresponding to the exp rule in the definition of SN, where $\mathsf{SN}\, M$ follows from $M \to_{\mathsf{SN}} N$ and $\mathsf{SN}\, N$. By ind. hyp. we have $M \to_{\mathsf{sn}} N$ and $\mathsf{sn}\, N$, and then we need a way of showing that the $\to_{\mathsf{sn}}$ relation is backward closed under sn. This is the goal of Lemma 4.8 below. In fact, what this lemma states is that $\to_{\mathsf{sn}}$ is also sound: if we have a $\to_{\mathsf{sn}}$ reduction from M to N and N is strongly normalizing, then M must also be. This reflects the idea behinds the $\to_{\mathsf{sn}}$ relation: it characterizes all computation strategies or possible reductions.
Coming back to Lemma 4.8, its proof goes by case analysis on the derivation of $M \to_{\mathsf{sn}} N$ plus three auxiliary results –one for each case (Lemmas 4.6, 3.3.3 and 4.7).

Next, we need to prove that $\mathsf{sn}\, N$ and $\mathsf{sn}\, M[x{:=}N]$ implies $\mathsf{sn}\, (\lambda x M)N$. However, –as in the previous lemma– since we are explicitly dealing with alpha-conversion, we need a stronger induction hypothesis, so we state the thesis is valid for any Q alpha-convertible with $M[x{:=}N]$:

Lemma 4.6 (Weak Head Expansion) *Let* $\mathsf{sn}\, N$, $\mathsf{sn}\, Q$ *and* $Q \sim_\alpha M[x{:=}N]$. *Then* $\mathsf{sn}\, (\lambda x M)N$.

Proof. By lexicographic induction on $\mathsf{sn}\, N$ and $\mathsf{sn}\, Q$. Assume $(\lambda x M)N \to_\beta P$ for some P and then proceed by cases proving $\mathsf{sn}\, P$:

- Case $P = M[x{:=}N]$. By Lemma 3.2 and hypothesis.
- Case $P = (\lambda x M')N$ with $\lambda x M \to_\beta \lambda x M'$, which in turn must follow from $M \to_\beta M'$. By Lemma 2.17, we have $M[x{:=}N] \to_\beta R \sim_\alpha M'[x{:=}N]$ for some R. Then, using Lemma 2.18 we obtain S such that $Q \to_\beta S \sim_\alpha R$. Hence, since $S \sim_\alpha M'[x{:=}N]$ by transitivity of $\sim_\alpha$, we obtain by ind. hyp. $\mathsf{sn}\,(\lambda x M')N$.
- Case $P = (\lambda x M)N'$, obtained from $N \to_\beta N'$. By Lemma 2.16.3, $M[x{:=}N] \twoheadrightarrow M[x{:=}N']$. Also, by Lemma 3.2 we get $\mathsf{sn}\,(M[x{:=}N])$. We apply Lemma 3.3.2 to obtain $\mathsf{sn}\,(M[x{:=}N'])$ and then, by ind. hyp. we conclude $\mathsf{sn}\,((\lambda x M)N')$. □

Next, we introduce the following lemma which will be useful later on Lemma 4.8:

Lemma 4.7 *Let $M \to_{\mathsf{sn}} M'$ as well as M, N and $M'N$ be* sn. *Then* $\mathsf{sn}\,MN$.

Proof. By lexicographic induction on $\mathsf{sn}\,M$, $\mathsf{sn}\,N$. Assume some $MN \to_\beta P$ and proceed by cases showing $\mathsf{sn}\,P$:

- Case $P = M''N$ with $M \to_\beta M''$. By Lemma 4.5, either $M' \sim_\alpha M''$ or there exists some Q such that $M' \twoheadrightarrow Q$ and $M'' \to_{\mathsf{sn}} Q$. The first case follows from the hypothesis $\mathsf{sn}\,(M'N)$ and Lemma 3.2. In the second case, first notice that by Lemma 3.3.2 and by Definition 2.14 we get $\mathsf{sn}\,(QN)$. Then, by definition of $\mathsf{sn}\,M$, we have $\mathsf{sn}\,M''$ and can apply the ind. hyp. to conclude $\mathsf{sn}\,P$.
- Case $P = MN'$ with $N \to_\beta N'$. Use then the ind. hyp. and $\mathsf{sn}\,(M'N')$. □

Lemma 4.8 (Backward Closure of sn) *Let $M \to_{\mathsf{sn}} N$ and* $\mathsf{sn}\,M$. *Then* $\mathsf{sn}\,M$.

Proof. By induction on $M \to_{\mathsf{sn}} N$.

- Case β: Using Lemma 4.6.
- Case appL: Assume $M \to_{\mathsf{sn}} M'$ and $\mathsf{sn}\,(M'N)$. By Lemma 3.3.3 we have $\mathsf{sn}\,M'$ and $\mathsf{sn}\,N$. By ind. hyp. we have $\mathsf{sn}\,M$, and using Lemma 4.7, we get $\mathsf{sn}\,(MN)$.
- Case α: Use Lemma 3.2 and the ind. hyp. □

Now we only need two preparatory results:

Lemma 4.9 (Soundness of SNe with respect to ne) $\mathsf{SNe}_x\,M \Rightarrow \mathsf{ne}_x\,M$.
Proof. By a simple induction on $\mathsf{SNe}_x\,M$. □

Lemma 4.10 $M \to_{\mathsf{SN}} N \Rightarrow M \twoheadrightarrow N$

Proof. By induction on $M \to_{\mathsf{SN}} N$.

- Case α: $M \to_{\mathsf{SN}} N$ is obtained from $M \to_{\mathsf{SN}} P \sim_\alpha N$. By ind. hyp. we have $M \twoheadrightarrow N$. Also, by Definition 2.14 we get $P \twoheadrightarrow N$. Then, by transitivity of $\sim_\alpha$ we have $M \twoheadrightarrow N$.
- Case β. By Definition 2.14.
- Case appL: $MN \to_{\mathsf{SN}} M'N$ is obtained from $M \to_{\mathsf{SN}} M'$. By ind. hyp. we have $M \twoheadrightarrow N$ hence, by Lemma 2.16.4 we have $MN \twoheadrightarrow M'N$. □

Finally, we prove that the alternative definition of the strongly normalizing terms is sound with respect to the accessible definition:

Theorem 4.11 (Soundness of SN)

1. $\mathsf{SN}\, M \Rightarrow \mathsf{sn}\, M$
2. $\mathsf{SNe}_x\, M \Rightarrow \mathsf{sn}\, M$
3. $M \to_{\mathsf{SN}} N \Rightarrow M \to_{\mathsf{sn}} N$

Proof. By simultaneous induction, following Definition 3.4.

- Case `v`: By Lemma 3.3.1.
- Case `app`: Necessarily $M = PQ$ and $\mathsf{SNe}_x\,(PQ)$ is obtained from $\mathsf{SNe}_x\, P$ and $\mathsf{SN}\, Q$. Then, by ind. hyp. we have $\mathsf{sn}\, P$ and $\mathsf{sn}\, Q$, and by Lemma 4.9 we get $\mathsf{ne}\, P$. Finally, by Lemma 4.3 we obtain $\mathsf{sn}\, PQ$.
- Case `ne`: By ind. hyp.
- Case λ: By ind. hyp. followed by Lemma 3.3.4.
- Case `exp`: By ind. hyp. followed by Lemma 4.8.
- Case β: By ind. hyp. and Definition 4.4.
- Case `appL`: By ind. hyp. and Definition 4.4. □

5 Strong Normalization Theorem

In this section we present the proof of the Strong Normalization Theorem for typable terms in the simply typed Lambda Calculus.

5.1 Typing judgements

Definition 5.1 *Types are defined by the following grammar, starting from a category τ of ground types:*

$$\alpha, \beta ::= \tau \mid \alpha \to \beta.$$

We introduce an ordering between types in order to later perform complete induction on them:

Definition 5.2 *We write $\alpha \preceq \beta$ when the type α is a (structural) component of type β, and $\alpha \prec \beta$ when it is a proper component.*

Definition 5.3 *The system of assignment of simple types to lambda terms is the following:*

$$\vdash \mathtt{v}: \frac{x \in \Gamma}{\Gamma \vdash x : \Gamma x} \qquad \vdash \mathtt{a}: \frac{\Gamma \vdash M : \alpha \to \beta \qquad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \beta} \qquad \vdash \lambda: \frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x M : \alpha \to \beta}$$

where Γ stands for a context (implemented as a finite list) of variable declarations of the form $x_1 : \alpha_1$, $x_2 : \alpha_2, \ldots, x_n : \alpha_n$. In rule $\vdash$ `v`, $x \in \Gamma$ means that x is declared in Γ, and Γx is the (last) type declared for x in Γ.
The following are proven in [3]:

Lemma 5.4 (Weakening) *Let $x \# M$ and $\Gamma \vdash M : \alpha$. Then $\Gamma, x : \beta \vdash M : \alpha$.*

Lemma 5.5 (Subject Reduction) $\Gamma \vdash M : \alpha \wedge M \twoheadrightarrow N \Rightarrow \Gamma \vdash N : \alpha$.

5.2 Typed Substitutions

Next, we introduce typed substitutions (w.r.t. the free variables of a term M):

Definition 5.6 *(σ, M) assigns to the variables in Γ terms of appropriate types under Δ —all of which is to be written $\sigma : \Gamma \to \Delta \downarrow M$— iff for all $x * M$ s.t. $x \in \Gamma$, we have $\Delta \vdash \sigma x : \Gamma x$.*

These are useful for stating that, if a term M is typable then $M\sigma$ is also typable, as the following lemma —proven in [3]— declares:

Lemma 5.7 *Let $\Gamma \vdash M : \alpha$ and $\sigma : \Gamma \to \Delta \downarrow M$. Then $\Delta \vdash M\sigma : \alpha$.*

Then, the following three lemmas –also proven in [3]– hold:

Lemma 5.8 *Let $x\#(\sigma, \lambda yM)$ and $\sigma : \Gamma \to \Delta \downarrow \lambda yM$.*
Then $(\sigma, y{:=}x) : (\Gamma, y{:}\alpha) \to (\Delta, x{:}\alpha) \downarrow M$.

Lemma 5.9 *Let $x\#M$ and $\sigma : \Gamma \to \Delta \downarrow M$.*
Then $(\sigma, x{:=}y) : (\Gamma, y{:}\alpha) \to (\Delta, x{:}\alpha) \downarrow M$.

Lemma 5.10 *Let $\Gamma \vdash M : \alpha$. Then $[x{:=}M] : (\Gamma, x{:}\alpha) \to \Gamma \downarrow M$.*

We end up this subsection with some useful general results.

Lemma 5.11 *Let $\mathsf{ne}_x\, M$ and $\Gamma \vdash x : \beta$. Then:*

1. $\Gamma \vdash M : \alpha \Rightarrow \alpha \preceq \beta$
2. $\Gamma \vdash M : \alpha \to \gamma \Rightarrow \alpha \prec \beta$

Proof. Immediate. □

Lemma 5.12 $u\#M \Rightarrow M\sigma = M(\sigma, u{:=}N)$

Proof. By definition of equality of restrictions (Def. 2.7). □

5.3 Strong Normalization

It is possible to prove by using a simple induction on the given type system that the typable terms are strongly normalizing . For this, we need to deal with the case of applications, that will be proved after the theorem (Lemma 5.14).

Theorem 5.13 (Strong Normalization) $\Gamma \vdash M : \alpha \Rightarrow \mathsf{SN}\, M$.

Proof. By induction on $\Gamma \vdash M : \alpha$.

- Case $\vdash$`v`: By Definition 3.4.
- Case $\vdash$`a`: $\Gamma \vdash MN : \alpha$ is obtained from $\Gamma \vdash M : \beta \to \alpha$ and $\Gamma \vdash N : \beta$. By ind. hyp. we get $\mathsf{SN}\, M$ and $\mathsf{SN}\, N$. Then, by Lemma 5.14 2 (ii) we have $\mathsf{SN}\, MN$.
- Case $\vdash\lambda$: By ind. hyp. and Definition 3.4. □

We may now turn our attention to the main lemma. As already said, its purpose is to establish that $\mathsf{SN}\, MN$ follows from $\mathsf{SN}\, M$ and $\mathsf{SN}\, N$ for typable M and N but, in order to obtain this, some additional results must be proven simultaneously.

Lemma 5.14 *Let* $\Gamma \vdash M : \alpha$, $\mathsf{SN}\, N$, $\sigma : \Gamma \to \Delta \downarrow M$, $\mathsf{Unary}\, \sigma\, x\, N$ *and* $\Gamma \vdash x : \beta$. *Then:*

1. $\mathsf{SNe}_y\, M \Rightarrow (i)\ \mathsf{SN}\, M\sigma$, $(ii)\ y{\neq}x \Rightarrow \mathsf{SNe}_{\sigma y}\, M\sigma$ *and* $(iii)\ \alpha = \beta{\to}\gamma \Rightarrow \mathsf{SN}\, MN$.
2. $\mathsf{SN}\, M \Rightarrow (i)\ \mathsf{SN}\, M\sigma$ *and* $(ii)\ \alpha = \beta{\to}\gamma \Rightarrow \mathsf{SN}\, MN$.
3. $M \to_{\mathsf{SN}} P \Rightarrow M\sigma \to_{\mathsf{SN}} P\sigma$.

Proof. The proof proceeds by complete induction on the structure of the type β, and subordinate induction on Definition 3.4 indexed by M. So, to begin with, take β and assume the statement for all of its proper components. We proceed to the subordinate induction:

- Case `v`: Then $M = y$ and we have to show 1: (i) $\mathsf{SN}\, y\sigma$, (ii) $y{\neq}x \Rightarrow \mathsf{SNe}_{\sigma y}\, y\sigma$ and (iii) $\alpha = \beta{\to}\gamma \Rightarrow \mathsf{SN}\, yN$. As to (i), either $y = x$ or not. If true, then by Def. 2.20 we have $\sigma y = N$. If not, —also by same definition— $\sigma y = z$ is a variable. Either case: $\mathsf{SN}\, \sigma y$. As to (ii), if $y{\neq}x$ then in addition by Def. 3.4 we have $\mathsf{SNe}_z\, z$. As to (iii): by rule `app` of Def. 3.4 on $\mathsf{SNe}_y\, y$ and $\mathsf{SN}\, N$, followed by rule `ne`.
- Case `app`: Then $M = PQ$ and $\mathsf{SNe}_y\, PQ$ follows from $\mathsf{SNe}_y\, P$ and $\mathsf{SN}\, Q$, and $\Gamma \vdash PQ : \alpha$ follows from $\Gamma \vdash P : \delta \to \alpha$ and $\Gamma \vdash Q : \delta$. We need to show 1: (i) $\mathsf{SN}\, (PQ)\sigma$, (ii) $y{\neq}x \Rightarrow \mathsf{SNe}_{\sigma y}\, (PQ)\sigma$ and (iii) $\alpha = \beta{\to}\gamma \Rightarrow \mathsf{SN}\, (PQ)N$. First, by ind. hyp. we have $\mathsf{SN}\, P\sigma$ and $\mathsf{SN}\, Q\sigma$. Besides, by Lem. 5.7 we get $\Delta \vdash P\sigma : \delta \to \alpha$ and $\Delta \vdash Q\sigma : \delta$. Now, we have either $y = x$ or not. In the first case, by Lem. 4.9 we have $\mathsf{ne}_x\, P$, thus by Lem. 5.11.2 it follows $\delta \prec \beta$. Then, as to (i), we can apply the main ind. hyp. 2(ii) with $M = P\sigma$, $N = Q\sigma$ and $\beta = \delta$ and get $\mathsf{SN}\, P\sigma Q\sigma$ which equals to $\mathsf{SN}\, (PQ)\sigma$, as desired. On the contrary, if $y{\neq}x$ then by ind. hyp. (ii) we get $\mathsf{SNe}_{\sigma y}\, P\sigma$. Further, by definition of SN it follows $\mathsf{SNe}_{\sigma y}\, P\sigma Q\sigma$, thus $\mathsf{SN}\, (PQ)\sigma$. (ii) Just proven. (iii) Similar to case `v`.
- Case `ne`: Then $\mathsf{SN}\, M$ follows from $\mathsf{SNe}_y\, M$ and 2 is immediate by the main ind. hyp. 1 (i) and (iii).
- Case λ: Then $M = \lambda yP$ and $\mathsf{SN}\, \lambda yP$ follows from $\mathsf{SN}\, P$ and $\Gamma \vdash \lambda yP : \delta \to \epsilon$ from $\Gamma, y : \delta \vdash P : \epsilon$. We need to show 2: (i) $\mathsf{SN}\, (\lambda yP)\sigma$ and (ii) $\mathsf{SN}\, (\lambda yP)N$. As to (i), we have $(\lambda yP)\sigma = \lambda zP(\sigma, y{:=}z)$ with $z = \chi(\sigma, \lambda xP)$. We now proceed by cases: either $y = x$ or not. In the first case, before we can apply ind. hyp. with $\mathsf{SN}\, P$ and $(\sigma, x{:=}z)$ we need to show $\Gamma, x : \delta \vdash x : \beta$. However, this does not necessarily follow, since δ is rather arbitrary. Nevertheless, we can proceed us follows: let us fix some $u = \chi(\iota, P)$. First, by Lemma 2.6.1 we have $u\#(\iota, P)$ and thus by Lemma 2.6.2 get $u\#P$. Now, by Lemma 2.6.1 we have $z\#\lambda xP$ and then by Lemma 5.8 obtain $(\sigma, x{:=}z) : \Gamma, x : \delta \to \Gamma, z : \delta \downarrow P$, thus by Lemma 5.9 have $((\sigma, x{:=}z), u{:=}v) : \Gamma, x{:}\delta, u{:}\beta \to \Delta, z{:}\delta, v{:}\beta \downarrow P$ for any v. Besides, by Lemma 5.4 we have $\Gamma, x{:}\delta, u{:}\beta \vdash P : \epsilon$ and then by Def. 5.3 we get $\Gamma, x{:}\delta, u{:}\beta \vdash u : \beta$. In addition, by Lemma 2.21.2 we have $\mathsf{Unary}\, (\sigma, x{:=}z)\, x\, z$, thus by Lemma 2.21.4 we get $\mathsf{Unary}\, ((\sigma, x{:=}z), u{:=}v)\, u\, v$. Now, since $\mathsf{SN}\, v$, we can apply the ind. hyp. and get $\mathsf{SN}\, P((\sigma, x{:=}z), u{:=}v)$. Then, by Lemma 5.12 $P((\sigma, x{:=}z), u{:=}v) = P(\sigma, x{:=}z)$, and thus $\mathsf{SN}\, P(\sigma, x{:=}z)$. Finally, by the λ rule of Def. 3.4 we obtain $\mathsf{SN}\, (\lambda xP)\sigma$, as desired. Now if, on the other hand, $y \neq x$, then by Lemma 2.21.3 we have $\mathsf{Unary}\, (\sigma, y{:=}z)\, x\, N$ and thus we can apply the ind. hyp. to obtain $\mathsf{SN}\, P(\sigma, y{:=}z)$

and therefore derive $\mathsf{SN}\,(\lambda yP)\sigma$ in the same way. As to (ii), first notice that, since $\alpha = \beta \to \gamma$, it follows $\beta = \delta$, hence $\Gamma, y{:}\beta \vdash y : \beta$. Then, by Lemma 2.21.1 we have $\mathsf{Unary}\,[y{:=}N]\,y\,N$ and by Lem. 5.10 we have $[y{:=}N] : \Gamma, y{:}\beta \to \Gamma \downarrow P$, thus we can apply the ind. hyp. (i) and obtain $\mathsf{SN}\,P[y{:=}N]$. Lastly, by β rule of Def. 3.4 we obtain $(\lambda yP)N \to_{\mathsf{SN}} P[y{:=}N]$ and thus by `exp` rule we can derive $\mathsf{SN}\,(\lambda yP)N$.

- Case `exp`: Then $\mathsf{SN}\,M$ follows from $M \to_{\mathsf{SN}} P$ and $\mathsf{SN}\,P$, and we need to show 2: (i) $\mathsf{SN}\,M\sigma$ and (ii) $\mathsf{SN}\,MN$. As to (i), by the main ind. hyp. 3 we have $M\sigma \to_{\mathsf{SN}} P\sigma$ and by ind. hyp. 2(i) $\mathsf{SN}\,P\sigma$, thus by rule `exp` we obtain $\mathsf{SN}\,M\sigma$. As to (ii), by ind. hyp. (ii) on $\mathsf{SN}\,P$ we have $\mathsf{SN}\,PN$, therefore by rule `appL` applied to $M \to_{\mathsf{SN}} P$ we get $MN \to_{\mathsf{SN}} PN$ and thus by `exp` we can derive $\mathsf{SN}\,MN$, as desired.
- Case β: Then $(\lambda xM)N \to_{\mathsf{SN}} M[x{:=}N]$ follows from $\mathsf{SN}\,N$ and we need to show 3: $((\lambda xM)N)\sigma \to_{\mathsf{SN}} M[x{:=}N]\sigma$. First, notice that by Def. 2.5 we have $((\lambda xM)N)\sigma = (\lambda zM(\sigma, x{:=}z))N\sigma$ with $z = \chi(\sigma, \lambda xM)$. In addition, by the main ind. hyp. 1(i) we have $\mathsf{SN}\,N\sigma$, thus by β rule of Def. 3.4 we have $(\lambda zM(\sigma, x{:=}z))N\sigma \to_{\mathsf{SN}} M(\sigma, x{:=}z)[z{:=}N\sigma]$. Furthermore, by Lem. 2.6.1 we get $z\#(\sigma, \lambda yM)$, hence by Cor. 2.12.1 we have $M(\sigma, x{:=}z)[z{:=}N\sigma] \sim_\alpha M[x{:=}N]\sigma$. Finally, we can use rule α of Def. 3.4 and obtain $((\lambda xM)N)\sigma \to_{\mathsf{SN}} M[x{:=}N]\sigma$.
- Case `appL`: Then $MN \to_{\mathsf{SN}} M'N$ follows from $M \to_{\mathsf{SN}} M'$ and we must show 3: $(MN)\sigma \to_{\mathsf{SN}} (M'N)\sigma$. By ind. hyp. we have $M\sigma \to_{\mathsf{SN}} M'\sigma$, thus by `appL` rule we can apply $N\sigma$ on both sides and obtain $M\sigma N\sigma \to_{\mathsf{SN}} M'\sigma N\sigma$ which by Def. 2.5 equals to $(MN)\sigma \to_{\mathsf{SN}} (M'N)\sigma$.
- Case α: Then $M \to_{\mathsf{SN}} P$ follows form $M \to_{\mathsf{SN}} N$ and $N \sim_\alpha P$ and we need to show 3: $M\sigma \to_{\mathsf{SN}} P\sigma$. By ind. hyp. we have $M\sigma \to_{\mathsf{SN}} N\sigma$ and by Lem. 2.11.8 we get $N\sigma = P\sigma$, thus by Lemma. 2.11.1 we obtain $N\sigma \sim_\alpha P\sigma$ and then by α rule of Def. 3.4 we can derive $M\sigma \to_{\mathsf{SN}} P\sigma$, as desired. □

6 Conclusions

The preceding work should be assessed in connection with the expectations stated in the Introduction. We assign relevance first of all to the complexity of the formalized version, both in Agda and in mathematical English.

Concerning the latter, we believe that this method makes it possible to come closer to a presentation in textbook style that does not hide details that need formalization. Of course, the main price paid has been the explicit handling of alpha-conversion; this has shown itself mainly in the formulation of the main definitions and results, which often must replace identity of terms by their alpha conversion, due to the renaming implicit in the effect of substitution on abstractions. As to additional, housekeeping lemmas, these seem to have been circumscribed to results of closure under alpha conversion. Also Lemma 5.12 is particularly noticeable. It could be replaced by a proof that Definition 3.4 is closed under alpha conversion.

Another aspect of the complexity has to do with the kind of proof methods required: when dealing with terms, we have been able to proceed by using only structural induction. In addition to this, we used of course induction on the various predicates and relations introduced, among which there is the somewhat complex mutual Def-

inition 3.4 of Section 3, as well as the well-founded induction on types which was the main method in the proof we chose to formalize.

On the other hand, the size of the Agda code has by no means exploded: it is of about 600 lines, split almost in halves between the proof of soundness of the inductive characterization of the strongly normalizing terms of Section 3 and the theorem of strong normalization property of Section 5. The complexity of the development can also be assessed by the effort required: the first author completed the work as part of his Master's thesis in about 400 hours of work. This of course reveals a quite high cost for each Agda line of code.

Our pre-existing library providing the basics for the formalization of the calculus by the method chosen was only marginally updated. The required new results are shown in Section 2. The library was readily understood and made use of by the author of the formalization, despite his not having much prior experience in Agda (this was of only one tutorial course).

We believe we have made progress towards developing a corpus of formalized metatheory of the Lambda Calculus somewhat along the lines of [9]. We also think that the present approach allows to aspire at achieving such a goal using the sort of explicit mathematical English we have used above, in turn supported by the Agda version.

This work should be compared to other formalizations of strong normalization, most notably those referred to in [1]. They employ variants of higher-order abstract syntax and of the de Bruijn notation, both of which of course dispense with alpha conversion. A full discussion is outside the scope of the present paper, but should be carried out. We should for that matter experiment with the full challenge posed in [1], using Kripke-style logical relations for proving strong normalization. The use of typed reduction will then force us to reformulate the calculus. We also have tried a formalization in Constructive Type Theory using Agda of principles of induction implementing the Barendregt variable convention –see [5,4], somewhat similarly to [12]. Again, we are still short of a full comparison in this respect.

References

[1] Abel, A., G. Allais, A. Hameer, B. Pientka, A. Momigliano, S. Schäfer and K. Stark, *POPLMark reloaded: Mechanizing proofs by logical relations*, Journal of Functional Programming **29** (2019).

[2] Copello, E., *Agda library for Formal metatheory of the lambda calculus using Stoughton's substitution*, Available at https://github.com/ernius/formalmetatheory-stoughton.

[3] Copello, E., N. Szasz and Álvaro Tasistro, *Formal metatheory of the lambda calculus using Stoughton's substitution*, Theoretical Computer Science **685** (2017), pp. 65 – 82.

[4] Copello, E., N. Szasz and Álvaro Tasistro, *Machine-checked Proof of the Church-Rosser Theorem for the Lambda Calculus Using the Barendregt Variable Convention in Constructive Type Theory*, Electronic Notes in Theoretical Computer Science **338** (2018), pp. 79 – 95, the 12th Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2017).
URL http://www.sciencedirect.com/science/article/pii/S1571066118300720

[5] Copello, E., Álvaro Tasistro, N. Szasz, A. Bove and M. Fernández, *Alpha-Structural Induction and Recursion for the Lambda Calculus in Constructive Type Theory*, Electronic Notes in Theoretical Computer Science **323** (2016), pp. 109 – 124.
URL http://www.sciencedirect.com/science/article/pii/S1571066116300354

[6] Copes, M., N. Szasz and A. Tasistro, *Formalization in Constructive Type Theory of the Standardization Theorem for the Lambda Calculus using Multiple Substitution*, in: F. Blanqui and G. Reis, editors, Proceedings of the 13th International Workshop on *Logical Frameworks and Meta-Languages: Theory and Practice,* Oxford, UK, 7th July 2018, Electronic Proceedings in Theoretical Computer Science **274** (2018), pp. 27–41.

[7] Curry, H. B. and R. Feys, "Combinatory Logic, Volume I," North-Holland, 1958, xvi+417 pp., second printing 1968.

[8] Joachimski, F. and R. Matthes, *Short Proofs of Normalization for the simply- typed λ-calculus, permutative conversions and Gödel's T*, Archive for Mathematical Logic **42** (2003), pp. 59–87.

[9] Nipkow, T. and S. Berghofer, *Fundamental Properties of Lambda-calculus* (2019), available from https://isabelle.in.tum.de/library/HOL/HOL-Proofs-Lambda/document.pdf.

[10] Norell, U., "Towards a Practical Programming Language Based on Dependent Type Theory," Ph.D. thesis, Department of Computer Science and Engineering, Chalmers University of Technology (2007).

[11] Stoughton, A., *Substitution revisited*, Theoretical Computer Science **59** (1988), pp. 317–325.

[12] Urban, C. and C. Tasson, *Nominal Techniques in Isabelle/HOL*, in: R. Nieuwenhuis, editor, *Automated Deduction – CADE-20*, Lecture Notes in Computer Science **3632**, Springer Berlin Heidelberg, 2005 pp. 38–53.

[13] van Raamsdonk, F. and P. Severi, *On Normalisation*, Technical report, Centrum voor Wiskunde en Informatica (CWI), Amsterdam (1995).