Formalization in Constructive Type Theory of the Standardization Theorem for the Lambda Calculus using Multiple Substitution

Martín Copes Nora Szasz Álvaro Tasistro
Universidad ORT Uruguay
{copes,szasz,tasistro}@ort.edu.uy

We present a full formalization in Martin-Löf's Constructive Type Theory of the Standardization Theorem for the Lambda Calculus using first-order syntax with one sort of names for both free and bound variables and Stoughton's multiple substitution. Our formalization is based on a proof by Ryo Kashima, in which a notion of β-reducibility with a standard sequence is captured by an inductive relation. The proof uses only structural induction over the syntax and the relations defined, which is possible due to the specific formulation of substitution that we employ. The whole development has been machine-checked using the system Agda.

1 Introduction

In [3] a formalization of the Lambda Calculus in Martin-Löf's Constructive Type Theory is presented, which uses first-order syntax with one sort of names for both free and bound variables that does not identify α-convertible terms, and a multiple substitution operation introduced by Stoughton in [11]. The approach enables the authors to prove in a completely formal and quite elegant way significant results about the metatheory of the Lambda Calculus, namely the Church-Rosser Theorem and Subject Reduction for the simply typed Lambda Calculus à la Curry. The authors developed a library [2] with definitions and lemmas for implementing and manipulating substitutions that was key for achieving the mentioned results, in particular by using only simple standard methods of structural induction on terms and reduction relations.

In the present work we extend the above mentioned metatheoretical study by proving the Standardization Theorem for β-reduction, which we further use to prove that the leftmost-outermost reduction strategy always finds the normal form of a term provided that it exists. The Standardization Theorem is a well-known result in the Lambda Calculus that was first proved by Curry and Feys in [5]. It states that if a term M β-reduces to a term N, then there exists a standard β-reduction sequence from M to N. A reduction sequence is considered standard if successive redexes are contracted from left to right (regarding the linear syntax) possibly with some jumps.

The proof hereby formalized is the one given by Ryo Kashima in [8] where the notion of β-reducibility with a standard sequence is captured by an inductive relation in very much the same way as minimal complete developments are captured by the so-called *parallel* reduction relation in e.g. Tait and Martin-Löf's method for proving the Church-Rosser theorem. This allows for an elegant inductive development as opposed to basing the proofs on notions like residuals and finite developments as in the classical proofs by Curry-Feys and Barendregt [5, 1].

All the definitions and proofs that appear in this article have been machine-checked with the system Agda [10]. In the subsequent text we will mix Agda code and (informal) proofs in English with a

F. Blanqui and G. Reis (Eds.): LFMTP'18
EPTCS 274, 2018, pp. 27–41, doi:10.4204/EPTCS.274.3

considerable level of detail so that they serve for clarifying their formalization. The complete code is available at `https://github.com/mcopes73/standardization-agda`.

In section 2 we present the basic concepts of the Lambda Calculus, together with some definitions and results from the library produced in [3] on which our work is based, as well as extensions thereof. In section 3 we present the proof of the Standardization Theorem. In section 4 we present the proof of the Leftmost Reduction Theorem for β-reduction. In section 5 we compare our development with other similar efforts in the literature, and present our overall conclusions.

2 Preliminaries

In what follows we will introduce the main definitions and results in [3, 2] that are previous to this work and are used in our formalization. We present the definitions directly using Agda code along with informal explanations, while the proofs are written in English to ease their reading. A certain degree of familiarity with the Agda syntax or at least with that of functional languages like Haskell is assumed.

We shall start by defining λ-terms using the same set of names for both bound and free variables. We use natural numbers to name variables for sake of concreteness.

```
V = ℕ
data Λ : Set where
  v    : V → Λ
  _·_  : Λ → Λ → Λ
  ƛ    : V → Λ → Λ
```

Agda is pretty liberal with regard to the naming of functions and the positions of their arguments. Notice the notation for declaring the infix application constructor, i.e. `_·_`. This underscore notation is extended to mixfix operators.

The classical notions of *free* and *fresh* (not free) variable in a term, which are denoted by $*$ and # respectively, are defined as binary relations between variables and terms in the usual way (we omit the definitions for reasons of space):

```
data _*_ : V → Λ → Set
data _#_ : V → Λ → Set
```

Substitutions are *identity-almost-everywhere* functions associating a term to every variable. We can generate every concrete substitution by starting up from the empty substitution ι that maps each variable to itself as a term, and employing the update operation $\prec\!+$, such that if σ is a substitution, then $\sigma \prec\!+ (x, M)$ is the substitution equal to σ everywhere except at x, where it yields M:

```
Σ = V → Λ

ι : Σ
ι = id ∘ v

_≺+_ : Σ → V × Λ → Σ
(σ ≺+ (x , M)) y with x ≟ y
... | yes _ = M
... | no  _ = σ y
```

Notice that in the definition of $\prec\!+$ we use the `with` construct, which allows us to perform pattern matching on the result of evaluating the expression `x` $\stackrel{?}{=}$ `y`. This expression decides the equality between

the variables x and y and has type Dec ≡, whose constructors are yes and no applied to the corresponding proof objects.

In general, we shall consider properties concerning the substitutions for the free variables of a term M, i.e. their *restrictions* to such variables. The type of restrictions R is defined as: R = Σ × Λ, and we note in the informal language such a restriction as $\sigma \restriction M$. This means that we are restricting the substitution σ to the free variables of M only. We will also use the following notion: $x \# (\sigma \restriction M)$, which stands for x *fresh in the* σ*-value of every free variable of* M:

```
_#↾_ : V → R → Set
x #↾ (σ , M) = (y : V) → y * M → x # (σ y)
```

The application of substitution σ to the term M is noted $M \bullet \sigma$, and it is defined by *structural recursion* on M. The fact that structural recursion is sufficient for stating this very concrete definition is a (very welcome) non-trivial consequence of the employment of multiple substitutions.

```
_•_ : Λ → Σ → Λ
(v x) • σ = σ x
(M · N) • σ = (M • σ) · (N • σ)
(ƛ x M) • σ = ƛ y (M • (σ ≺+ (x , v y)))
     where y = χ (σ , ƛ x M)
```

Notice the last line of the definition: when performing a substitution over an abstraction, the bound variable x is always replaced with a new one. This new variable y is obtained by means of a choice function χ, such that $\chi(\sigma, M) \# (\sigma \restriction M)$. In this way, y does not capture any of the names introduced into its scope by effect of the substitution[1]. When reasoning with substitutions, this uniform renaming of bound variables allows us to avoid case analyses; it also has other nice consequences, to be noticed shortly. For the sake of readability, we define the single substitution of a term N for a variable x in M with the traditional notation $M[x := N]$.

```
_[_:=_] : Λ -> V -> Λ -> Λ
M [ x := N ] = M • (ι ≺+ (x , N))
```

Alpha-conversion ($\sim_\alpha$) is defined as the following inductive binary relation on terms:

```
data _∼α_ : Λ → Λ → Set where
  ∼v : {x : V} → (v x) ∼α (v x)
  ∼· : {M M' N N' : Λ} → M ∼α M' → N ∼α N' → M · N ∼α M' · N'
  ∼ƛ : {M M' : Λ}{x x' y : V} → y # ƛ x M → y # ƛ x' M'
                              → M [x := v y] ∼α M' [x' := v y]
                              → ƛ x M ∼α ƛ x' M'
```

Arguments to a function declared between braces { } are optional and in subsequent applications of the function in question they are inferred by the type-checker. The first two constructors above implement the classical rules for variables and application. The last constructor states that two abstractions are α-convertible if and only if their bodies are α-convertible after replacing the bound variables with a common fresh name. From this definition it follows that $\sim_\alpha$ is an equivalence relation, as shown in [3]. As it is the case in [11], α-equivalent terms become identical when submitted to the same substitution. This is due to the fact that abstractions are uniformly renamed, and that the new name chosen by the χ function is determined only by the restriction of the substitution to the free variables of the terms, which is the same one if the terms are α-equivalent. This is proven in [3], and we just mention the corresponding lemma here:

[1] In fact, χ is implemented by just finding the first variable not free in the given restriction.

```
lemmaM∼M'→Mσ≡M'σ : {M M' : Λ}{σ : Σ} → M ∼α M' → M • σ ≡ M' • σ
```

From now on we present definitions and results not included in the library [2].

Firstly, we have proven that this definition of alpha equivalence is decidable:

```
_∼α?_ : ∀ A B -> Dec (A ∼α B)
```

Given a binary relation $\rightsquigarrow$, we define its α-reflexive-transitive closure as follows:

```
data α-star (⇝ : Rel) : Rel where
    refl : ∀{M} → α-star ⇝ M M
    α-step : ∀{M N N'} → α-star ⇝ M N' → N' ∼α N → α-star ⇝ M N
    append : ∀ {M N K} → α-star ⇝ M K → ⇝ K N → α-star ⇝ M N
```

where `Rel` is the type of binary relations over terms.
This is the kind of closure that will be applied to our one-step reduction relations. It represents sequences of $\rightsquigarrow$ steps allowing α conversions, which have to be made explicit because we are dealing with concrete terms, i.e. terms not identified under α conversion. In informal notation we shall write the α-reflexive-transitive closure of a relation with the classical two-headed arrow. From the definition given we can easily prove that, for any relation $\rightsquigarrow$, $M \rightsquigarrow N$ implies $M \rightsquigarrow\!\!\!\!\rightarrow N$, and that $\rightsquigarrow\!\!\!\!\rightarrow$ is transitive. The first proof is straightforward using the constructors `append` and `refl`. Transitivity is proven by induction on the definition of `α-star`. Therefore we have, in Agda:

```
α-star-singl : ∀{⇝ M N} -> ⇝ M N -> α-star ⇝ M N
α-star-trans : ∀{⇝ M N K} -> α-star ⇝ M K -> α-star ⇝ K N -> α-star ⇝ M N
```

Following Kashima [8], we define β-contraction taking into account the position where the contracted redex appears in the term relative to the other redexes. We start by defining two auxiliary functions: `isAbs` is a predicate that decides whether a term is an abstraction and `countRedexes` a function that counts the number of β-redexes in a term.

```
data isAbs : Λ -> Set where
  abs : forall {x M} -> isAbs (ƛ x M)
```

We need to prove that `isAbs` is decidable before being able to define `countRedexes`, since the number of redexes for an application depends on whether the left term is an abstraction. The proof is straightforward:

```
isAbs? : (M : Λ) -> Dec (isAbs M)
```

Using this property we can define `countRedexes` as follows:

```
countRedexes : Λ -> ℕ
countRedexes (v _) = 0
countRedexes (M · N) with isAbs? M
... | yes _ = suc (countRedexes M + countRedexes N)
... | no _ = countRedexes M + countRedexes N
countRedexes (ƛ _ M) = countRedexes M
```

Considering the linear syntax of terms, redexes will be numbered in a left-to-right fashion, starting from zero. We shall start by defining the *contraction of the n-th redex* as a relation between terms depending on the natural number n.

```
data _β_@_ : Λ -> Λ -> ℕ -> Set where
  outer-redex : ∀ {x A B} -> ((ƛ x A) · B) β (A [ x := B ]) @ 0
  appNoAbsL : ∀ {n A B C} -> A β B @ n -> ¬ isAbs A -> (A · C) β (B · C) @ n
  appAbsL : ∀ {n A B C} -> A β B @ n -> isAbs A -> (A · C) β (B · C) @ (suc n)
  appNoAbsR : ∀ {n A B C} -> A β B @ n -> ¬ isAbs C
              -> (C · A) β (C · B) @ (n + countRedexes C)
  appAbsR : ∀ {n A B C} -> A β B @ n -> isAbs C
              -> (C · A) β (C · B) @ (suc (n + countRedexes C))
  abs : ∀ {n x A B} -> A β B @ n -> (ƛ x A) β (ƛ x B) @ n
```

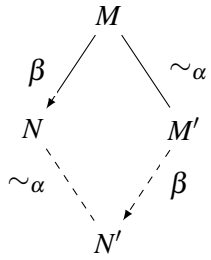
The `outer-redex` constructor allows the contraction of the outermost redex, numbered as the one at position zero. The next four constructors are used to perform contractions inside applications. In order to determine the number of the redex contracted we need to identify whether the left hand side term of the application is an abstraction or not (which is necessary to know whether we are stepping over a redex to reduce an inner one). Finally, the `abs` constructor allows contractions inside an abstraction.

One-step β-*reduction* ($\longrightarrow_\beta$) from M to N can now be defined as the existence of a natural number n such that N can be obtained by contracting the n-th redex from M. We use Agda's dependent ordered pair constructor Σ to express existential quantification.

```
_⟶β_ : Λ -> Λ -> Set
M ⟶β N = Σ ℕ (\n -> M β N @ n)
```

It is easily proven by structural induction that this definition is equivalent to the compatible (with the syntactic constructors) closure of ordinary β-contraction.

One interesting result that will be useful in our development is the following α-β compatibility property:



which we state in Agda as the following lemma:

```
lem-βα : ∀{M N M'} -> M ⟶β N -> M ∼α M'-> Σ Λ (λ N' -> (M' ⟶β N') ∧ (N' ∼α N))
```

We finally introduce β-reduction $\twoheadrightarrow_\beta$ as the α-reflexive-transitive closure of the contraction $\longrightarrow_\beta$:

```
_↠β_ : Λ -> Λ -> Set
_↠β_ = α-star (_⟶β_)
```

3 The Standardization Theorem

In the present section we show the formalization of the Standardization Theorem in Constructive Type Theory that follows the proof given by Kashima in [8]. For the sake of clarity, some lemmas are presented in a different order than the one proposed by Kashima. Nonetheless, the formalized results and definitions are the same unless otherwise stated.

3.1 Standard Reduction Sequences

A *reduction sequence* is a sequence of terms $M_0, M_1, ..., M_n$ such that M_{i+1} is obtained from M_i by the contraction of some redex, i.e., $(\forall\, i \in 0...n-1)\ M_i \longrightarrow_\beta M_{i+1}$. We call a reduction sequence *standard* if and only if subsequent steps are non decreasing in the number of the redex contracted. Kashima defines a standard beta reduction sequence as: If $M_0 \xrightarrow{n_1}_\beta M_1 \xrightarrow{n_2}_\beta \ ... \ \xrightarrow{n_k}_\beta M_k$ then $n_1 \leq n_2 \leq ... \leq n_k$, where $M \xrightarrow{n}_\beta N$ is the β-contraction of the n-th redex in M, which we note $M\,\beta\,N\,@\,n$ in our development.

We implement this notion in Agda by defining a relation indexed on a natural number that keeps track of the lower bound allowed for the next redex to be contracted.

```
data seqβ-st (M : Λ) : (N : Λ) -> ℕ -> Set where
  nil : seqβ-st M M 0
  α-step : ∀ {n K N} -> seqβ-st M K n -> K ∼α N -> seqβ-st M N n
  β-step : ∀ {K n n₀ N} -> seqβ-st M K n -> K β N @ n₀ -> n₀ ≥ n -> seqβ-st M N n₀
```

The relation is reflexive and allows for α-steps, which do not appear explicitly in Kashima's definition because the latter relies on the (informal) syntactic identification of α convertible terms. The "three dots" of Kashimas's sequence $M_0, M_1, ..., M_k$ are implemented as follows: we can append a term to the reduction sequence provided that it can be obtained by the contraction of a redex at a position greater than or equal to the current lower bound. Using this relation the Standardization Theorem can be precisely stated as the existence of a standard sequence between two terms among which there is a β-reduction:

```
standardization : ∀{M N} -> M ↠β N -> Σ ℕ (λ n -> seqβ-st M N n)
```

3.2 Two Useful Reduction Relations

The next step is to capture the existence of a standard sequence as an inductively defined reduction relation between terms. To this end, Kashima introduces two auxiliary one-step reduction relations:

$\longrightarrow_l$ stands for *leftmost reduction* and corresponds to the contraction of the leftmost redex, i.e. the one at position zero:

```
data _⟶l_ : Λ -> Λ -> Set
    M ⟶l N =  M β N @ 0
```

$\longrightarrow_{hap}$ stands for *head contraction in application* and represents the contraction of the redex in the head position of a chain of applications, i.e.: $(\lambda x M_0)M_1M_2...M_n \longrightarrow_{hap} M_0[x := M_1]\ M_2...M_n$. We define this relation in Agda as follows:

```
data _⟶hap_ : Λ -> Λ -> Set where
  hap-head : ∀{x A B} -> (ƛ x A) · B ⟶hap (A [ x := B ])
  hap-chain : ∀{C A B} -> A ⟶hap B -> (A · C) ⟶hap (B · C)
```

Now $\twoheadrightarrow_l$ and $\twoheadrightarrow_{hap}$ are defined as the α-reflexive-transitive closures of $\longrightarrow_l$ and $\longrightarrow_{hap}$ respectively.

```
_↠hap_ : Λ -> Λ -> Set
_↠hap_ = α-star (_⟶hap_)

_↠l_ : Λ -> Λ -> Set
_↠l_ = α-star (_⟶l_)
```

The first two lemmas state that head reduction in application $\twoheadrightarrow_{hap}$ is compatible with application and substitution respectively.

```
hap-app-r : ∀{M N P} -> M ↠hap N -> M · P ↠hap N · P
```

Proof. By induction on the definition of $M \twoheadrightarrow_{hap} N$.

- Case `refl`: We have to prove $(M\ P) \twoheadrightarrow_{hap} (M\ P)$, which follows by `refl`.
- Case α`-step`: Assume that $M \twoheadrightarrow_{hap} N$ follows from $M \twoheadrightarrow_{hap} N'$ and $N' \sim_\alpha N$. Then, we obtain $M\ P \twoheadrightarrow_{hap} N'\ P$ from the induction hypothesis, and since $N'\ P \sim_\alpha N\ P$, we construct our goal using `st-alpha`.
- Case `append`: Assume $M \twoheadrightarrow_{hap} N$ follows from $M \twoheadrightarrow_{hap} K$ and $K \longrightarrow_{hap} N$. Then we can obtain $M\ P \twoheadrightarrow_{hap} K\ P$ from the induction hypothesis and $K\ P \longrightarrow_{hap} N\ P$ from rule `hap-chain` applied to $K \longrightarrow_{hap} N$. From these, we construct our goal using `append`. □

In order to prove that substitution preserves the head reduction relation, we need two lemmas from the substitution library [2]. The first one states that substituting y for x and then N for y yields a result α-equivalent to substituting N for x, provided y is fresh enough. The second one is a form of the substitution composition lemma:

```
corollary1SubstLemma : {x y : V} {σ : Σ}{M N : Λ} → y #↓ (σ , ƛ x M)
    → ((M • (σ ≺+ (x , v y))) [y := N]) ∼α  (M • (σ ≺+ (x , N)))

corollary1Prop7 : {M N : Λ}{σ : Σ}{x : V}
    → M • (σ ≺+ (x , N • σ)) ≡ (M [x := N]) • σ
```

Now we prove that substitution preserves $\longrightarrow_{hap}$ up to $\sim_\alpha$:

```
lem-hap-subst : ∀{σ M N} -> M ⟶hap N
    ->  Σ Λ (λ N' -> ((M • σ) ⟶hap N') ∧ (N' ∼α (N • σ)))
```

Proof. By induction on the definition of $M \longrightarrow_{hap} N$

- Case `hap-head`: We want to prove that $((\lambda x\ A)\ B) \bullet \sigma \longrightarrow_{hap} N \wedge N \sim_\alpha (A[x := B]) \bullet \sigma$, for some term N. Starting from the left hand side:
  $((\lambda x\ A)\ B) \bullet \sigma$
  $\equiv$ (Def. $\bullet$)
  $(\lambda y\ A \bullet (\sigma \prec\!+(x,y)))\ (B \bullet \sigma)$ where $y = \chi(\sigma, \lambda x\ A)$
  $\longrightarrow_{hap}$ (`hap-head`)
  $(A \bullet (\sigma \prec\!+(x,y)))\ [y := B \bullet \sigma]$
  $\sim_\alpha$ (`corollary1substLemma`, $y \#\ (\sigma, \lambda x\ A)$)
  $A \bullet (\sigma \prec\!+(x,\ B \bullet \sigma))$
  $\equiv$ (`corollary1Prop7`)
  $(A\ [x := B]) \bullet \sigma$
- Case `hap-chain`: We need to prove that there exists a term K such that $(M\ P) \bullet \sigma \longrightarrow_{hap} K \wedge K \sim_\alpha (N\ P) \bullet \sigma$, assuming $M \bullet \sigma \longrightarrow_{hap} N \bullet \sigma$. This follows directly from rule `hap-chain` applied to $M \bullet \sigma \longrightarrow_{hap} N \bullet \sigma$ and the definition of $\bullet$. □

Kashima originally formulates the previous result just for single substitutions, i.e., of the form $[x := P]$. Our result using multiple substitutions will allow us to rely only on structural induction in our proofs, as we shall see later. We can easily extend the previous result to $\twoheadrightarrow_{hap}$:

```
hap-subst : ∀{M N σ} -> M →→hap N -> (M • σ) →→hap (N • σ)
```

Proof. By induction on $M \twoheadrightarrow_{hap} N$:

- Case `refl`: Direct using `refl`.

- Case α-step: Assume $M \twoheadrightarrow_{hap} N'$ and $N' \sim_\alpha N$. Then, we obtain $M \bullet \sigma \twoheadrightarrow_{hap} N' \bullet \sigma$ from the induction hypothesis, and by lemmaM∼M'→Mσ≡M'σ mentioned in Section 2, we have that $N' \bullet \sigma \equiv N \bullet \sigma$, so we construct our goal using the α-step rule, since $\sim_\alpha$ is reflexive.
- Case append: Assume $M \twoheadrightarrow_{hap} N$ follows from $M \twoheadrightarrow_{hap} K$ and $K \longrightarrow_{hap} N$.
  Then we can obtain $M \bullet \sigma \twoheadrightarrow_{hap} K \bullet \sigma$ from the induction hypothesis and $(\exists N')(K \bullet \sigma \longrightarrow_{hap} N' \wedge N' \sim_\alpha N \bullet \sigma)$ from the previous lemma (lem-hap-subst) applied to $K \longrightarrow_{hap} N$. From this, using α-star-single we construct $K \bullet \sigma \twoheadrightarrow_{hap} N'$ and since $N' \sim_\alpha N$ we obtain $K \bullet \sigma \twoheadrightarrow_{hap} N$ from rule α-step. Finally, we prove our goal from the transitivity of $\twoheadrightarrow_{hap}$. □

Finally, notice that head reduction in application implies leftmost reduction:

```
lem-hap→l : ∀ {M N} -> M ⟶hap N -> M ⟶l N
```

And therefore the same inclusion holds for their α-reflexive-transitive closures:

```
hap→l : ∀{M N} -> M →→hap N -> M →→l N
```

3.3 Standard Reduction

Using $\twoheadrightarrow_{hap}$, Kashima characterizes the existence of a standard sequence as a further reduction relation $\twoheadrightarrow_{st}$, which stands for *standard reduction*, as follows:

```
data _→→st_ (L : Λ) : Λ -> Set where
  st-var : ∀{x} -> L →→hap (v x) -> L →→st (v x)
  st-app : ∀{A B C D} -> L →→hap (A · B) -> A →→st C -> B →→st D -> L →→st (C · D)
  st-abs : ∀{x A B} -> L →→hap (ƛ x A) -> A →→st B -> L →→st (ƛ x B)
  st-alpha : ∀{A' A} -> L →→st A' -> A' ∼α A -> L →→st A
```

The intention behind this relation is to characterize standard reduction sequences inductively. This definition allows us to perform as many $\twoheadrightarrow_{hap}$ steps as we want. After that, if we reach a variable, then we are done since we cannot do any more reductions (st-var). If the term is an application $A\ B$, then we can continue performing standard reductions on A and then on B (st-app). Finally, if the term is an abstraction, we can continue performing standard reductions inside the body of the abstraction (st-abs). Note that we are not forced to reduce all of the redexes that we encounter; given a redex, we can still apply st-app while skipping the head reduction. The last constructor (st-alpha) allows us to perform α-conversion. The preceding explanation may have shown that *standard reductions* correspond to *standard sequences* of reductions, i.e. that the former relation is included in the latter one. This is enough for proving the standardization theorem, as will be shown in the next subsection. We have further proven that actually the characterization of standard reduction sequences by the relation of standard reduction is complete, i.e. that the converse inclusion aldo holds.

The notion of standard reduction can be extended to substitutions. We say that *substitution* σ *standard-reduces to* σ' $(\sigma \twoheadrightarrow_{st} \sigma')$ if and only if for all variables x, $\sigma\ x \twoheadrightarrow_{st} \sigma'\ x$.

```
_→st_ : Σ → Σ → Set
σ →st σ' = (x : V) → σ x →→st σ' x
```

Reflexivity of $\twoheadrightarrow_{st}$ is proven by a direct induction on $M : \Lambda$.

```
st-refl : ∀{M} -> M →→st M
```

Appending head reductions in applications at the beginning of a standard reduction results in a standard reduction.

```
hap-st→st : ∀{L M N} -> L →→hap M -> M →→st N -> L →→st N
```

Proof. By induction on the definition of $M \twoheadrightarrow_{st} N$.

- Case `st-var`: We know that $M \twoheadrightarrow_{hap} x$. From this, $L \twoheadrightarrow_{hap} M$ and the transitivity of $\twoheadrightarrow_{hap}$ we conclude that $L \twoheadrightarrow_{hap} x$, and then $L \twoheadrightarrow_{st} x$ follows from `st-var`.
- For the `st-app` case, assume $M \twoheadrightarrow_{hap} A\ B$, $A \twoheadrightarrow_{st} C$ and $B \twoheadrightarrow_{st} D$. From $L \twoheadrightarrow_{hap} M$ and $M \twoheadrightarrow_{hap} A\ B$ we conclude that $L \twoheadrightarrow_{hap} A\ B$ by transitivity of $\twoheadrightarrow_{hap}$. Finally, from this plus $A \twoheadrightarrow_{st} C$ and $B \twoheadrightarrow_{st} D$, we conclude that $L \twoheadrightarrow_{st} C\ D$ using `st-app`.
- For the `st-abs` case, we assume $M \twoheadrightarrow_{hap} \lambda x\ A$ and $A \twoheadrightarrow_{st} B$. Similarly to the preceding case, we conclude that $L \twoheadrightarrow_{hap} \lambda x\ A$ from $L \twoheadrightarrow_{hap} M$, $M \twoheadrightarrow_{hap} \lambda x\ A$ and the transitivity of $\twoheadrightarrow_{hap}$. From this and $A \twoheadrightarrow_{st} B$, we conclude $L \twoheadrightarrow_{st} \lambda x\ B$ using `st-abs`.
- For the `st-alpha` case, we assume $M \twoheadrightarrow_{st} A'$ and $A' \sim_\alpha A$. We use constructor `st-alpha` applied to the induction hypothesis $L \twoheadrightarrow_{st} A'$ and $A' \sim_\alpha A'$ to complete our goal. □

We can now use the preceding lemma to prove that substitution is preserved by the $\twoheadrightarrow_{st}$ relation. This lemma is key to the proof and was originally stated by Kashima for single substitutions as: $M \twoheadrightarrow_{st} N$ and $P \twoheadrightarrow_{st} Q \Longrightarrow M[z := P] \twoheadrightarrow_{st} N[z := Q]$. By taking the substitution to be an arbitrary (multiple) σ instead of the particular case where we replace just one variable z, we obtain a definition of substitution by structural recursion, and hence we can prove this result using just structural induction (see [3] for a detailed explanation). The substitution lemma is then stated as follows:

```
st-substσ≅σ' : ∀{M N σ σ'} -> M →→st N -> σ →→st σ' -> M • σ →→st N • σ'
```

Proof. By induction on the definition of $M \twoheadrightarrow_{st} N$

- Case `st-var`: We have to prove $M \bullet \sigma \twoheadrightarrow_{st} x \bullet \sigma'$ under the hypotheses $M \twoheadrightarrow_{hap} x$ and $\sigma \rightarrow_{st} \sigma'$. From `hap-subst` applied to $M \twoheadrightarrow_{hap} x$ we know that $M \bullet \sigma \twoheadrightarrow_{hap} x \bullet \sigma$ and from the definition of $\sigma \rightarrow_{st} \sigma'$ we get that $x \bullet \sigma \twoheadrightarrow_{st} x \bullet \sigma'$. Therefore, from $M \bullet \sigma \twoheadrightarrow_{hap} x \bullet \sigma \twoheadrightarrow_{st} x \bullet \sigma'$ we conclude that $M \bullet \sigma \twoheadrightarrow_{st} x \bullet \sigma'$ using `hap-st→st`.
- Case `st-app`: Assume $M \twoheadrightarrow_{hap} A\ B$, $\sigma \rightarrow_{st} \sigma'$, $A \bullet \sigma \twoheadrightarrow_{st} C \bullet \sigma'$ and $B \bullet \sigma \twoheadrightarrow_{st} D \bullet \sigma'$. We have to prove $M \bullet \sigma \twoheadrightarrow_{st} (C\ D) \bullet \sigma'$. Now:
  $M \twoheadrightarrow_{hap} A\ B$
  $\Longrightarrow$ (`hap-subst`)
  $M \bullet \sigma \twoheadrightarrow_{hap} (A\ B) \bullet \sigma$
  $\equiv$ (Def. $\bullet$)
  $M \bullet \sigma \twoheadrightarrow_{hap} (A \bullet \sigma)\ (B \bullet \sigma)$
  $\Longrightarrow$ (`st-app` and hypothesis)
  $M \bullet \sigma \twoheadrightarrow_{st} (C \bullet \sigma')\ (D \bullet \sigma')$
  $\equiv$ (Def. $\bullet$)
  $M \bullet \sigma \twoheadrightarrow_{st} (C\ D) \bullet \sigma'$.
- Case `st-abs`: Assume $M \twoheadrightarrow_{hap} \lambda x\ A$, $A \twoheadrightarrow_{st} B$ and $\sigma \rightarrow_{st} \sigma'$. We prove $M \bullet \sigma \twoheadrightarrow_{st} (\lambda x\ B) \bullet \sigma'$:
  $M \twoheadrightarrow_{hap} \lambda x\ A$
  $\Longrightarrow$ (`hap-subst`)
  $M \bullet \sigma \twoheadrightarrow_{hap} (\lambda x\ A) \bullet \sigma$
  $\equiv$ (Def $\bullet$)
  $M \bullet \sigma \twoheadrightarrow_{hap} \lambda y_A\ (A \bullet \sigma \prec\!+\!(x,\ y_A))$ where $y_A = \chi(\sigma, \lambda x\ A)$ (1).

Let $z = \chi(\iota \downarrow ((\lambda x\, A) \bullet \sigma)\ ((\lambda x\, B) \bullet \sigma'))$. Due to the definition of the choice function χ, z is fresh in every term and substitution involved. We can now prove that:
$\lambda y_A\ (A \bullet \sigma \prec\!+ (x,\ y_A)) \sim_\alpha \lambda z\ (A \bullet \sigma \prec\!+ (x,\ z))$
$\Longrightarrow$ (`hap-`α and (1))
$M \bullet \sigma \twoheadrightarrow_{hap} \lambda z\ (A \bullet \sigma \prec\!+ (x,\ z))$ (2).
Note that by using multiple substitutions, our induction hypothesis is strong enough to allow us to use it with any pair of substitutions σ, σ' as long as $\sigma \to_{st} \sigma'$. Therefore, we can extract the following induction hypothesis from $A \twoheadrightarrow_{st} B$:
$A \bullet \sigma \prec\!+ (x,\ z) \twoheadrightarrow_{st} B \bullet \sigma' \prec\!+ (x,\ z)$ (3).
We can prove that $\sigma \prec\!+ (x,\ z) \to_{st} \sigma' \prec\!+ (x,\ z)$ because the $\twoheadrightarrow_{st}$ relation is reflexive (lemma `st-refl`), so replacing x for z in both substitutions will preserve the $\to_{st}$ relation. So, from (2), (3) and constructor `st-abs` we obtain that $M \bullet \sigma \twoheadrightarrow_{st} \lambda z\ (B \bullet \sigma' \prec\!+ (x,\ z))$ and we obtain our thesis using `st-alpha`, since $\lambda z\ (B \bullet \sigma' \prec\!+ (x,\ z)) \sim_\alpha (\lambda x\, B) \bullet \sigma'$, .

- Case `st-alpha`: We assume that $M \twoheadrightarrow_{st} N'$ and $N' \sim_\alpha N$ and want to prove that $M \bullet \sigma \twoheadrightarrow_{st} N \bullet \sigma'$. From the induction hypothesis we get that $M \bullet \sigma \twoheadrightarrow_{st} N' \bullet \sigma'$. In addition, we know that $N' \bullet \sigma' \sim_\alpha N \bullet \sigma'$, since they are equal (`lemmaM∼M'→Mσ≡M'σ`) and $\sim_\alpha$ is reflexive. From these we obtain our goal using the `st-alpha` rule. □

The following lemma states that if there is a standard reduction to a term that is a redex $(\lambda x.M)\ N$, then it is possible to construct a standard reduction to the contractum $M[x := N]$, somehow "inserting" the contraction in a right place:

```
st-abs-subst : ∀ {L M N x} -> L →→st (ƛ x M) · N -> L →→st (M [ x := N ])
```

Proof. From $L \twoheadrightarrow_{st} (\lambda x\, M)\ N$ and the definition of $\twoheadrightarrow_{st}$ we know that $L \twoheadrightarrow_{hap} P\ N'$ for some P and N' such that $P \twoheadrightarrow_{st} (\lambda x\, M)$ and $N' \twoheadrightarrow_{st} N$. In addition, from $P \twoheadrightarrow_{st} (\lambda x\, M)$ we know that $P \twoheadrightarrow_{hap} (\lambda x\, M')$ for some M' such that $M' \twoheadrightarrow_{st} M$. Then,
$P \twoheadrightarrow_{hap} (\lambda x\, M')$
$\Longrightarrow$ (`hap-app-r`)
$P\ N' \twoheadrightarrow_{hap} (\lambda x\, M')\ N'$ (1).
On the other hand,
$(\lambda x\, M')\ N'$
$\twoheadrightarrow_{hap}$ (α`-star-singl` applied to constructor `hap-head`)
$M'\ [x := N']$
$\twoheadrightarrow_{st}$ (`st-substσ≅σ'` with $M' \twoheadrightarrow_{st} M$ and $N' \twoheadrightarrow_{st} N$)
$M\ [x := N]$ (2).
From $L \twoheadrightarrow_{hap} P\ N'$, (1), (2) and the transitivity of $\twoheadrightarrow_{hap}$ we get that $L \twoheadrightarrow_{hap} M'\ [x := N']$, and since $M'\ [x := N'] \twoheadrightarrow_{st} M\ [x := N]$ we conclude that $L \twoheadrightarrow_{st} M\ [x := N]$ using lemma `hap-st→st`. □

Using this result, we can now prove that any β-contraction can be also inserted into a standard reduction:

```
st-β→st : ∀{L M N} -> L →→st M -> M ⟶β N -> L →→st N
```

Proof. By induction on $M \longrightarrow_\beta N$.

- The case `outer-redex` follows directly from the previous lemma `st-abs-subst`.
- All the application cases are solved by simply using `st-app` applied to the induction hypotheses. For example, if $M \longrightarrow_\beta N$ was constructed using the rule `appAbsL` then we know that $M = A\ C$, $N = B\ C$ and $(A\ C)\ \beta\ (B\ C)$ @ $(suc\ n)$, with $A\ \beta\ B$ @ n for some n. Since M is an application, $L \twoheadrightarrow_{st} M$

must have been constructed using either the `st-app` constructor or the `st-alpha` constructor. We will deal with all the `st-alpha` cases uniformly at the end, so let us focus on the `st-app` case for now. We know that $L \twoheadrightarrow_{hap} A'\ C'$, $A' \twoheadrightarrow_{st} A$ and $C' \twoheadrightarrow_{st} C$. We want to prove that $L \twoheadrightarrow_{st} B\ C$. From $A' \twoheadrightarrow_{st} A$ and $A \longrightarrow_{\beta} B$ we get that $A' \twoheadrightarrow_{st} B$ by the induction hypothesis. Finally, we prove this case using the `st-app` rule applied to $L \twoheadrightarrow_{hap} A'\ C'$, $A' \twoheadrightarrow_{st} B$ and $C' \twoheadrightarrow_{st} C$. The proofs for the other three application cases follow the same structure.

- The `abs` case also follows a similar pattern. We know that $\lambda xA \longrightarrow_{\beta} \lambda xB$ where $A \longrightarrow_{\beta} B$. Therefore, considering that $L \twoheadrightarrow_{st} \lambda xA$ was constructed using the `st-abs` rule, we have that $L \twoheadrightarrow_{hap} \lambda xA'$ and $A' \twoheadrightarrow_{st} A$ for some A'. The induction hypothesis applied to $A' \twoheadrightarrow_{st} A$ and $A \longrightarrow_{\beta} B$ gives us that $A' \twoheadrightarrow_{st} B$, and we obtain our goal $L \twoheadrightarrow_{st} \lambda xB$ using the `st-abs` rule applied to $L \twoheadrightarrow_{hap} \lambda xA'$ and $A' \longrightarrow_{\beta} B$.
- In all the previous cases we ignored the case where `L` $\twoheadrightarrow_{st}$`M` was constructed using the `st-alpha` constructor since we can prove this uniformly for all cases. We know that $L \twoheadrightarrow_{st} M'$ and $M' \sim_{\alpha} M$. In order to use the induction hypothesis we would need to have that $M' \longrightarrow_{\beta} K$ for some K. Since we know that $M \longrightarrow_{\beta} N$ and $M' \sim_{\alpha} M$ we can use the α-β diamond property of Section 2 (`lem-`$\beta\alpha$), to obtain a term K such that $M' \longrightarrow_{\beta} K$ and $K \sim_{\alpha} N$, so we prove our goal using the `st-alpha` rule. □

Finally, using this last result we can prove that if there is a sequence of β-reductions from M to N, then there is also a standard reduction between those two terms. The proof is a direct induction on $M \twoheadrightarrow_{\beta} N$:

```
β→st : ∀{M N} -> M →→β N -> M →→st N
```

3.4 Standard Sequences

The next results show the relation between the reduction relations $\twoheadrightarrow_{l}$, $\twoheadrightarrow_{hap}$ and $\twoheadrightarrow_{st}$ with the existence of a standard reduction sequence. Firstly notice that, since leftmost reductions always involve the reduction of redexes at position 0, then any sequence of leftmost reductions is a standard reduction sequence with lower bound 0.

```
nf→leftmost→seqβst : ∀{M N} -> M →→l N -> seqβ-st M N 0
```

As a corollary of this lemma and the fact that $M \twoheadrightarrow_{hap} N$ implies $M \twoheadrightarrow_{l} N$ (`lem-hap→l`), we obtain that if $M \twoheadrightarrow_{hap} N$, then there is a standard reduction sequence from M to N with lower bound 0:

```
hap→seqβst : ∀{M N} -> M →→hap N -> seqβ-st M N 0
```

The next result about *seqβ-st* will be useful to prove the subsequent lemma.

```
abs-seq : ∀ {x M N n} -> seqβ-st M N n -> seqβ-st (ƛ x M) (ƛ x N) n
```

Proof. We proceed by induction on the definition of *seqβ-st M N n*.

- Case `nil`: We know that *seqβ-st M M* 0 and therefore we construct our goal, *seqβ-st* (λxM) (λxM) 0, using `nil`.
- Case α`-step`: We know that *seqβ-st M K n* for some K such that $K \sim_{\alpha} N$. From the induction hypothesis we get that *seqβ-st* (λxM) (λxK) n and since $K \sim_{\alpha} N$ we can easily prove that $\lambda xK \sim_{\alpha} \lambda xN$. From this we prove the case using the `st-alpha` constructor.
- Case β`-step`: We know that *seqβ-st M K n* for some K such that $K\,\beta\,N\,@\,m$ with $n \leq m$. Similarly to the last case, the induction hypothesis tells us that *seqβ-st* (λxM) (λxK) n and from $K\,\beta\,N\,@\,m$ we can construct $(\lambda xK)\,\beta\,(\lambda xN)\,@\,m$ using rule `abs`. Finally, we prove our goal using constructor β`-step`. □

As for the $\twoheadrightarrow_{st}$ relation, if $M \twoheadrightarrow_{st} N$ then there is a standard reduction sequence from M to N, which we code in Agda as the existence of a lower bound for a standard reduction sequence:

```
st→seqβst : ∀{M N} -> M →→st N -> Σ ℕ (\n -> seqβ-st M N n)
```

Proof. By induction on the definition of $M \twoheadrightarrow_{st} N$.

- The case `st-var` can be easily proven using lemma `hap→seqβst`: since $M \twoheadrightarrow_{hap} x$, then there is a standard reduction sequence (with lower bound 0) from M to x.
- Similarly, the case `st-abs` also relies in this lemma, and the induction hypothesis: we know from `hap→seqβst` that there is a standard reduction sequence with lower bound 0 from M to $\lambda x.A$; the induction hypothesis tells us that there exists a natural number n such that there is a reduction sequence from A to B with lower bound n. Therefore, using lemma `abs-seq` we conclude that there must be a standard reduction sequence from M to B with lower bound n since $0 \leq n$.
- The case for `st-app` is slightly trickier since the lower bound that exists depends on certain characteristics of the terms involved: If $M \twoheadrightarrow_{st} N$ was constructed using the constructor `st-app`, that means that for some terms A, B , C and D: (1) $M \twoheadrightarrow_{hap} (A\ B)$, (2) $A \twoheadrightarrow_{st} C$ and (3) $B \twoheadrightarrow_{st} D$. We need to prove that there is a standard reduction sequence from M to $(C\ D)$. Using the induction hypotheses, let m and n be the lower bounds for the standard reduction sequences from A to C and from B to D respectively:
  1. If C is not an abstraction, and $B \sim_\alpha D$[2], then the lower bound for the standard reduction sequence will be m.
  2. If C is not an abstraction, and $B \not\sim_\alpha D$, then the lower bound for the standard reduction sequence will be $n +$ `countRedexes` C.
  3. If C is an abstraction, and $B \not\sim_\alpha D$, then the lower bound for the standard reduction sequence will be $suc\,(n +$ `countRedexes` $C\,)$.
  4. If C is an abstraction, and $B \sim_\alpha D$, then we need to do some further case analysis using the following lemma:
     ```
     lem-seq-appACBC-abs : ∀ {A C B n} -> seqβ-st A C n -> isAbs C
       -> (seqβ-st (A · B) (C · B) n) ∨ (seqβ-st (A · B) (C · B) (suc n))
     ```
     Note that if a reduction sequence ends in an abstraction, by appending the same application (or an α-equivalent one) to all of the terms in the sequence, the lower bound will remain the same if and only the abstraction is generated in the last beta step of the sequence and therefore does not affect the redex count in β-reductions. However, if the abstraction appears before that, then the lower bound of the reduction sequence must be increased by one, since a new redex at position 0 is formed. From this lemma we conclude that the lower bound must be either m or $suc\ m$ for this case.
- Finally, for the `st-alpha` case, we have that $M \twoheadrightarrow_{st} N'$ and $N' \sim_\alpha N$ and we want to prove the existence of a standard reduction sequence from M to N. The induction hypothesis gives us a standard reduction sequence from M to N' and we can directly perform an alpha step to N by using the α`-step` constructor from `seqβ-st`. □

The Standardization Theorem finally follows from this last result and lemma β`→st` that states that $M \twoheadrightarrow_\beta N$ implies $M \twoheadrightarrow_{st} N$:

```
standardization : ∀{M N} -> M →→β N -> Σ ℕ (λ n -> seqβ-st M N n)
standardization M→→βN = st→seqβst (β→st M→→βN)
```

[2]This is a possible scenario, since $\twoheadrightarrow_{st}$ includes $\sim_\alpha$.

4 The Leftmost Reduction Theorem

A quite relevant corollary of the Standardization Theorem is the Leftmost Reduction Theorem, which states that if a term M has a normal form, then the leftmost-outermost reduction strategy will find it. In the present section we show how this property can be derived from standardization. It is worth noticing that this proof was developed as part of the present work and is not present in Kashima's article.

We can directly characterize a term in normal form as one without redexes using the `countRedexes` function from section 2:

```
nf : Λ -> Set
nf M = countRedexes M ≡ 0
```

and now we can state the aforementioned property as the following lemma:

```
leftmost-nf : ∀{M N} -> M →→β N -> nf N -> M →→l N
```

In order to prove this result, we must first consider some lemmas. The first one states that the number of redexes of two α-equivalent terms is the same, which easily follows by induction on $M \sim_\alpha N$:

```
α→sameRedexCount : ∀ {M N} -> M ∼α N -> countRedexes M ≡ countRedexes N
```

The second lemma states that if a term M β-reduces to a term N in normal form, then the contracted redex must be the leftmost redex of M, i.e., the one at position zero:

```
nf→l : ∀{M N n} -> M β N @ n -> nf N -> n ≡ 0
```

Proof. We proceed by induction on $M\,\beta\,N\,@\,n$

- Case `outer-redex`: we have that $(\lambda xA)\,B\,\beta\,B[x := A]\,@\,0$. Our goal follows directly since rule `outer-redex` contracts the redex at position 0.
- Case `appNoAbsL`: we have that $(A\,C)\,\beta\,(B\,C)\,@\,n$ where $A\,\beta\,B\,@\,n$, A is not an abstraction and $(B\,C)$ is in normal form. From this, we know that `countRedexes` B + `countRedexes` $C \equiv 0$, and therefore `countRedexes` $B \equiv$ `0` which allows us to use the induction hypothesis with $A\,\beta\,B\,@\,n$ and conclude that $n \equiv 0$.
- Case `appAbsL`: we have that $(A\,C)\,\beta\,(B\,C)\,@\,(suc\,n)$ where $A\,\beta\,B\,@\,n$, A is an abstraction and $(B\,C)$ is in normal form. Since $(B\,C)$ is in normal form B cannot be an abstraction, but this is a contradiction since $A\,\beta\,B\,@\,n$ and A is an abstraction because contracting a redex in an abstraction always results in an abstraction (rule `abs`).
- Case `appNoAbsR`: we have that $(C\,A)\,\beta\,(C\,B)\,@\,(\texttt{countRedexes}\,C + n)$ where $A\,\beta\,B\,@\,n$, C is not an abstraction and $(C\;B)$ is in normal form. From this we know that `countRedexes` $C \equiv 0$ and `countRedexes` $B \equiv 0$. From the induction hypothesis we have that $n \equiv 0$, and since `countRedexes` $C \equiv 0$, $n +$ `countRedexes` $C \equiv 0$.
- Case `appAbsR`: we have that $(C\,A)\,\beta\,(C\,B)\,@\,suc\,(\texttt{countRedexes}\,C + n)$ where $A\,\beta\,B\,@\,n$, C is an abstraction and $(C\;B)$ is in normal form. However, this is a contradiction since $(C\;B)$ cannot be in normal form if C is an abstraction.
- Case `abs`: we have that $\lambda xA\ \beta\ \lambda xB\ @\ n$ where $A\ \beta\ B\ @\ n$ and λxB is in normal form. From this we know that B must be in normal form too and we can call the induction hypothesis for $A\ \beta\ B\ @\ n$, concluding that $n \equiv 0$. □

Finally, notice that a standard sequence with lower bound 0 must be a sequence of leftmost reductions, since all of the β-steps must involve the contraction of the redex at position 0, i.e. a leftmost reduction.

```
seqβ0→l : ∀ {A B} -> seqβ-st A B 0 -> A →→l B
```

The proof follows by a direct induction on *seqβ-st A B* 0.

Lets now turn our attention to the main lemma:

```
seqst→l : ∀{M N n} -> seqβ-st M N n -> nf N -> M →→l N
```

Proof. We proceed by induction on the definition of *seqβ-st M N n*.

- Case `nil`: We have that *seqβ-st A A* 0 and we need to prove that $A \twoheadrightarrow_l A$, which follows by constructor `refl`.
- Case α`-step`: We have that *seqβ-st A* B' *n* with $B' \sim_\alpha B$ and `nf` *B*. From this, we have that `countRedexes` $B' \equiv 0$ by lemma $\alpha\rightarrow$`sameRedexCount` and therefore, using the induction hypothesis we obtain `A` $\twoheadrightarrow_l$ `B'`. Finally, we construct our goal using rule α`-step`.
- Case β`-step`: We have *seqβ-st A* B' *n* and $B' \,\beta\, B \,@\, n'$, where $n \leq n'$ and `countRedexes` $B \equiv 0$. Using lemma `nf`$\rightarrow$`l` we have that $n' \equiv 0$, and so $n \equiv 0$. We then apply lemma `seqβ0`$\rightarrow$`l` to *seqβ-st A* B' 0 and get that $A \twoheadrightarrow_l B'$. Note that since $n' \equiv 0$, we also have that $B' \longrightarrow_l B$. Finally, from $A \twoheadrightarrow_l B'$ and $B' \longrightarrow_l B$ we conclude that $A \twoheadrightarrow_l B$ using rule `append`. □

Finally, if we have that $M \twoheadrightarrow_\beta N$, the Standardization Theorem lets us conclude that there exists a standard reduction sequence from M to N. Therefore, the desired property follows directly combining this result and the previous lemma:

```
leftmost-nf : ∀{M N} -> M →→β N -> nf N -> M →→l N
leftmost-nf M→→βN crN≡0 = seqst→l (proj2 (standardization M→→βN)) crN≡0
```

5 Conclusions

In this work we have extended some metatheoretical results from [3] by formalizing a proof of the Standardization Theorem in Lambda Calculus using Constructive Type Theory. We use a concrete approach to λ-terms and the notion of multiple substitution. The latter enables us to proceed by structural induction only, producing proofs that are easy to follow, yet fully formal. This work has also served to showcase the usefulness of the library produced in [2] and its suitability for the formalization of other metatheoretical properties of the Lambda Calculus. It is worth highlighting that the definitions and lemmas used to handle syntax and substitutions did not need to be modified or extended in any way and could be rapidly put into use by a programmer with a minimal training in Agda, namely the first author while working on his Master's thesis [4]. The Agda code reported in this paper is 890 lines long.

Other efforts to formalize Kashima's proof in the literature include one by Guidi in Matita [7] and another one by Vyšniauskas and Emerich in Coq [6]. However, what sets our development apart from these efforts is the use of a concrete syntax with names and our definition of multiple substitution. While this allows us to prove our lemmas using a clean structural inductive argument, they use a nameless syntax based on de Brujin indexes which results in some inductions being done on the size of the λ-terms. Another effort worth highlighting is that of McKinna and Pollack, who formalized a proof of the Standardization Theorem due to Takahashi [12] using the LEGO proof assistant [9].

Within the chosen syntax approach, we have to consider the work by Vestergaard and Brotherston [13] which uses modified rules of α-conversion and β-reduction based on unary substitution to formally prove the Church-Rosser theorem in Isabelle-HOL. Substitution does not proceed in cases of capture and they

use explicit α-conversion to perform the renaming achieved by our substitution. As a consequence, their development requires an administrative layer of reasoning for showing that α-conversion and β-reduction interact correctly. This consists in a rather complex definition of a new auxiliary relation for α-conversion, which we do not need.

In addition to proving the Standardization Theorem, Kashima proves a few other interesting results which could be a good follow up to the present work, e.g. the *Quasi-Leftmost Reduction Theorem*. An infinite β-reduction sequence is called quasi-leftmost if it contains infinitely many leftmost reduction steps $\longrightarrow_l$. As a corollary of the Standardization Theorem it can be proved that if M has a β-normal form, then there is no infinite quasi-leftmost β-reduction sequence from M.

References

[1] Hendrik Barendregt (1984): *The Lambda Calculus Its Syntax and Semantics*, revised edition. *Studies in Logic and the Foundations of Mathematics* 103, North Holland, doi:10.2307/2274112.

[2] Ernesto Copello: *Agda Library for Formal metatheory of the Lambda Calculus using Stoughton's substitution*. Available at https://github.com/ernius/formalmetatheory-stoughton.

[3] Ernesto Copello, Nora Szasz & Álvaro Tasistro (2017): *Formal metatheory of the Lambda Calculus using Stoughton's substitution*. *Theoretical Computer Science* 685, pp. 65 – 82, doi:10.1016/j.tcs.2016.08.025.

[4] Martin Copes (2018): *A machine checked proof of the Standardization Theorem in Lambda Calculus using multiple substitution*. Master's thesis, Universidad ORT Uruguay.

[5] H. B. Curry & R. Feys (1958): *Combinatory Logic, Volume I*. North-Holland. Second printing 1968.

[6] Johannes Emerich & Ignas Vyšniauskas (2014): *Coq formalisation of Postponement and Standardization theorems in the untyped lambda-calculus*. Available at https://github.com/knuton/la-girafe-sportive. ILLC, Universiteit van Amsterdam.

[7] Ferruccio Guidi (2012): *Standardization and Confluence in Pure Lambda-Calculus Formalized for the Matita Theorem Prover*. *Journal of Formalized Reasoning* 5(1), pp. 1–25, doi:10.6092/issn.1972-5787/3392. Available at `https://jfr.unibo.it/article/view/3392`.

[8] R. Kashima (2000): *A Proof of the Standardization Theorem in Lambda-Calculus*. Technical Report, Tokyo Institute of Technology. Department of Information Sciences. Available at `http://www.is.titech.ac.jp/~kashima/pub/C-145.pdf`.

[9] James McKinna & Robert Pollack (1999): *Some Lambda Calculus and Type Theory Formalized*. *Journal of Automated Reasoning* 23(3), pp. 373–409, doi:10.1023/A:1006294005493.

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

[11] A. Stoughton (1988): *Substitution Revisited*. *Theoretical Computer Science* 59, pp. 317–325. Available at `http://dx.doi.org/10.1016/0304-3975(88)90149-1`.

[12] M. Takahashi (1995): *Parallel Reductions in λ-Calculus*. *Information and Computation* 118(1), pp. 120 – 127, doi:10.1006/inco.1995.1057. Available at `http://www.sciencedirect.com/science/article/pii/S0890540185710577`.

[13] René Vestergaard & James Brotherston (2003): *A Formalised First-Order Confluence Proof for the λ-Calculus using One-Sorted Variable Names*. *Information and Computation* 183(2), pp. 212–244, doi:10.1016/S0890-5401(03)00023-3.