Hereditary Substitution for the $\lambda\Delta$-Calculus

Harley Eades and Aaron Stump

Department of Computer Science
University of Iowa

Hereditary substitution is a form of type-bounded iterated substitution, first made explicit by Watkins et al. and Adams in order to show normalization of proof terms for various constructive logics. This paper is the first to apply hereditary substitution to show normalization of a type theory corresponding to a non-constructive logic, namely the $\lambda\Delta$-calculus as formulated by Rehof. We show that there is a non-trivial extension of the hereditary substitution function of the simply-typed λ-calculus to one for the $\lambda\Delta$-calculus. Then hereditary substitution is used to prove normalization.

1 Introduction

In 1992 M. Parigot defined an algorithmic interpretation of classical natural deduction called the $\lambda\mu$-calculus [15]. His original theory consisted of complete sequents for types which often made the theory difficult to reason about, especially when one wished to adapt any well-known results of intuitionistic type theory to his classical type theory. Later, in 1994 N. Rehof and M. Sørensen defined the $\lambda\Delta$-calculus which is provably equivalent to the $\lambda\mu$-calculus [1]. Due to their equivalence, any results obtained for the $\lambda\mu$-calculus apply to the $\lambda\Delta$-calculus by translation. Now the $\lambda\Delta$-calculus is essentially an extension of the simply typed λ-calculus (STLC), hence adapting results from intuitionistic type theory to the $\lambda\Delta$-calculus is less complicated. The main result of this paper is the adaptation of a well-known proof technique for showing normalization of intuitionistic typed λ-calculi, called hereditary substitution, to the $\lambda\Delta$-calculus. We stress that proving normalization of the $\lambda\Delta$-calculus is not our contribution. This is already well known [7, 16]. In fact it is strongly normalizing. The adaptation of the proof method to the $\lambda\Delta$-calculus is however our main contribution.

The central idea behind the hereditary substitution proof method is to prove normalization using a lexicographic combination of an ordering on types and the strict subexpression ordering on proofs. This central idea has been used in normalization proofs dating all the way back to Prawitz in 1965. Since then it has been used to show normalization for many simply-typed λ-calculi [4, 9, 11, 14]. Extracting the constructive content from these proofs one will obtain a function much like capture avoiding substitution, except when a redex which was not present in the input is created as a result of substitution, that redex is recursively reduced. This substitution function is called hereditary substitution. It was first made explicit by K. Watkins et al. in [18] for non-dependent types and R. Adams in [3] for dependent types. In previous work, the authors showed how to apply the hereditary-substitution method to prove normalization of Stratified System F (SSF), a type theory of predicative polymorphism studied by Leivant [8, 13].

The motivation for using the hereditary substitution method over other well-known methods for showing normalization is that it is simpler. It provides a directly defined substitution which preserves normal forms. Its definition is essentially a combination of the reduction relation with capture avoiding substitution. It has found important application in logical frameworks based on canonical forms [18].

[1] By this we mean that everything provable in the $\lambda\mu$-calculus is provable in the $\lambda\Delta$-calculus, but β-reduction is not step-by-step equivalent.

Submitted to:
COS 2013

In these frameworks hereditary substitution replaces ordinary capture avoiding substitution in order to maintain canonicity.

We begin with defining the λΔ-calculus and presenting some basic meta-results in Sect. 2 and Sect. 3. Then we give the definition of the hereditary substitution function for the simply typed λ-calculus in Sect. 4. In Sect. 5.2 we extend the definition of the hereditary substitution function for the simply typed λ-calculus with a new function called the structural hereditary substitution function. Then its correctness properties are presented in Sect. 5.3. The hereditary substitution function is then used to conclude normalization of the λΔ-calculus in Sect. 6. We conclude with a related work section in Sect. 7.

2 The λΔ-Calculus

The λΔ-calculus is a straightforward extension of the simply typed λ-calculus. The syntax is defined in Figure 1. The type b is an arbitrary base type. Negation is defined as it is in intuitionistic type theory, that

$$\begin{array}{rrcl}
\text{(Types)} & T,A,B,C & ::= & \bot \mid b \mid A \to B \\
\text{(Terms)} & t & ::= & x \mid \lambda x:T.t \mid \Delta x:T.t \mid t_1\, t_2 \\
\text{(Normal Forms)} & n,m & ::= & x \mid \lambda x:T.n \mid \Delta x:T.n \mid h\, n \\
\text{(Heads)} & h & ::= & x \mid h\, n \\
\text{(Contexts)} & \Gamma & ::= & \cdot \mid x:A \mid \Gamma_1,\Gamma_2
\end{array}$$

Figure 1: Syntax for Types and Terms

is, $\neg A =^{def} A \to \bot$, where $\bot$ is absurdity. Arbitrary syntactically defined normal forms will be denoted by the meta-variables n and m, and arbitrary typing contexts will be denoted by the meta-variable Γ. We assume at all times that all variables in the domain of Γ are unique. In addition we rearrange the objects in Γ freely without indication.

The typing rules are defined in Figure 2. The operational semantics are the compatible closure of the rules in Figure 3. It is easy to see based on the typing rules that the Δ-abstraction is the introduction

$$\frac{}{\Gamma,x:A \vdash x:A}\ \textsc{Ax} \qquad \frac{\Gamma,x:A \vdash t:B}{\Gamma \vdash \lambda x:A.t:A \to B}\ \textsc{Lam}$$

$$\frac{\Gamma \vdash t_2:A \quad \Gamma \vdash t_1:A \to B}{\Gamma \vdash t_1\, t_2:B}\ \textsc{App} \qquad \frac{\Gamma,x:\neg A \vdash t:\bot}{\Gamma \vdash \Delta x:\neg A.t:A}\ \textsc{Delta}$$

Figure 2: Typing Rules

form for double negation. We annotate the Δ-abstraction with the type of the bound variable to make the definition of the hereditary substitution function a little less complicated. Removing this annotation should not cause any significant problems. On a more programmatic front the Δ-abstraction is a control operator. It can simulate Felleisen's control operators; see [17] for more information on this. N. Rehof and M. Sørensen also extend the operational semantics with a structural reduction rule for the Δ-abstraction (StructRed in Figure 3). This rule is called structural because it does not amount to a computational step, rather pushes the application into the body of the Δ-abstraction potentially creating additional redexes. We denote the reflexive and transitive closure of $\rightsquigarrow$ as $\rightsquigarrow^*$. We also define $t \rightsquigarrow^! t'$ to

$$\frac{}{(\lambda x : T.t)\, t' \rightsquigarrow [t'/x]t} \quad \textsc{Beta}$$

$$\frac{\begin{array}{c} y \text{ fresh in } t \text{ and } t' \\ z \text{ fresh in } t \text{ and } t' \end{array}}{(\Delta x : \neg(T_1 \to T_2).t)\, t' \rightsquigarrow \Delta y : \neg T_2.[\lambda z : T_1 \to T_2.(y\,(z\,t'))/x]t} \quad \textsc{StructRed}$$

Figure 3: Operational Semantics

mean that $t \rightsquigarrow^* t'$ and t' is normal. Now that we have defined the $\lambda\Delta$-calculus we state several well-known meta-results that will be needed throughout the sequel.

3 Basic Syntactic Lemmas

The following meta-results are well-known so we omit their proofs. We do not always explicitly state the use of these results. The first two properties are weakening and substitution for the typing relation.

Lemma 1 (Weakening for Typing)**.** *If $\Gamma \vdash t : T$ then $\Gamma, x : T' \vdash t : T$ for any fresh variable x and type T'.*

Proof. Straightforward induction on the assumed typing derivation. □

Lemma 2 (Substitution for Typing)**.** *If $\Gamma \vdash t : T$ and $\Gamma, x : T, \Gamma' \vdash t' : T'$ then $\Gamma \vdash [t/x]t' : T'$.*

Proof. Straightforward induction on the second assumed typing derivation. □

The final three properties are, confluence, type preservation and inversion of the typing relation. The proof of the confluence and type preservation can be found in [17] and the proof of the latter is trivial.

Theorem 3 (Confluence)**.** *If $t_1 \rightsquigarrow^* t_2$ and $t_1 \rightsquigarrow^* t_3$, then there exists a term t_4, such that, $t_2 \rightsquigarrow^* t_4$ and $t_3 \rightsquigarrow^* t_4$.*

Theorem 4 (Preservation)**.** *If $\Gamma \vdash t : T$ and $t \rightsquigarrow t'$ then $\Gamma \vdash t' : T$.*

Theorem 5 (Inversion)**.**

i. *If $\Gamma \vdash x : T$ then $x \in \Gamma$.*

ii. *If $\Gamma \vdash \lambda x : T_1.t : T_1 \to T_2$ then $\Gamma, x : T_1 \vdash t : T_2$.*

iii. *If $\Gamma \vdash \Delta x : \neg T.t : T$ then $\Gamma, x : \neg T \vdash t : \bot$.*

Proof. This can be shown by straightforward induction on the assumed typing derivations. □

At this point we have everything we need to state and prove correct the hereditary substitution function.

4 The Hereditary Substitution Function for STLC

In the introduction we gave an informal definition of the hereditary substitution function. It is exactly like capture-avoiding substitution, except that if any redexes are introduced as a result of substitution those redexes are recursively reduced. In fact hereditary substitution in general does not modify any redexes already in the input. However, if there are no redexes present in the input then the output of the hereditary substitution function will not have any either. This is one of the main correctness properties of the hereditary substitution function.

The definition of the hereditary substitution function strongly depends on the existence of an ordering on types. In fact if no such ordering exists then it is unclear if the hereditary substitution can be defined and proved correct. We say "unclear" here because it is not known if there exists a means of proving the hereditary substitution function correct without an ordering on types. We conjecture that one may be able to give some semantic interpretation of hereditary substitution and show correctness with respect to the semantics. However, this is just a conjecture. Fortunately, a very simple ordering exists on the types of STLC and the λΔ-calculus.

Definition 6. *We define an ordering on types T as the compatible closure of the following formulas.*

$$\begin{array}{rcl} A \to B & > & A \\ A \to B & > & B \end{array}$$

The ordering defined above is simply the strict subexpression ordering on types where the absurdity and base types are minimal elements. This ordering is clearly well founded.

The definition of the hereditary substitution function depends on being able to detect when a new redex has been created as a result of substitution. When a new redex is created it must be able to also detect that the ordering on types has decreased. To detect both of these situations the hereditary substitution function uses the following partial function.

Definition 7. *We define the partial function* ctype *which computes the type of an application in head normal form. It is defined as follows:*

$$\mathsf{ctype}_T(x,x) = T$$

$$\mathsf{ctype}_T(x,t_1\,t_2) = T''$$
$$\quad \textit{Where } \mathsf{ctype}_T(x,t_1) = T' \to T''.$$

The following lemma list the most important results about the ctype function.

Lemma 8 (Properties of ctype)**.**

i. If $\mathsf{ctype}_T(x,t) = T'$ then $\mathsf{head}(t) = x$ and $T' \leq T$.

ii. If $\Gamma, x : T, \Gamma' \vdash t : T'$ and $\mathsf{ctype}_T(x,t) = T''$ then $T' \equiv T''$.

Proof. Both cases can be shown by straightforward induction on the structure of t. The proof can be found in Appendix A.1. □

We now have everything we need to state the hereditary substitution function for STLC. We denote the hereditary substitution function by $[t/x]^A t'$ where A is the type of x and is called the cut type, due to the correspondence between hereditary substitution and cut elimination. In the definition of the hereditary substitution function it is assumed that all variables are renamed as to prevent variable capture. It is also defined with respect to the termination metric (A,t) in lexicographic combination of our ordering on types and the strict subexpression ordering on terms.

Definition 9. *The hereditary substitution function is defined as follows:*

$$[t/x]^A x = t$$

$$[t/x]^A y = y$$

$$[t/x]^A(\lambda y : A'.t') = \lambda y : A'.([t/x]^A t')$$

$$[t/x]^A(t_1\, t_2) = ([t/x]^A t_1)\,([t/x]^A t_2)$$
Where $([t/x]^A t_1)$ *is not a* λ*-abstraction, or both* $([t/x]^A t_1)$ *and* t_1 *are* λ*-abstractions.*

$$[t/x]^A(t_1\, t_2) = [s_2'/y]^{A''} s_1'$$
Where $([t/x]^A t_1) = \lambda y : A''.s_1'$ *for some y,* s_1' *and* A''*,* $[t/x]^A t_2 = s_2'$*, and* $\mathsf{ctype}_A(x, t_1) = A'' \to A'$*.*

The definition of the hereditary substitution function is similar to the definition of capture-avoiding substitution. The differences show up in the cases for application. The last case of the hereditary substitution function handles the case when a new β-redex is created as a result of substitution. This case depends heavily on the following lemma.

Lemma 10 (Properties of ctype Continued)**.** *If* $\Gamma, x : T, \Gamma' \vdash t_1\, t_2 : T'$*,* $\Gamma \vdash t : T$*,* $[t/x]^T t_1 = \lambda y : T_1.t'$*, and* t_1 *is not a* λ*-abstraction, then* t_1 *is in head normal form and there exists a type A such that* $\mathsf{ctype}_T(x, t_1) = A$*.*

Proof. This can be shown by induction on the structure of $t_1\, t_2$. See part one of the proof in Appendix A.2. □

The previous properties state that if we have created a redex using hereditary substitution, then ctype must be defined. This in turn tells us that in the case where hereditary substitution is applied to a term of the form $t_1\, t_2$ and a new β-redex is created then the head of t_1 must be the variable being replaced. Furthermore, recursively applying the hereditary substitution function to t_1 must yield a λ-abstraction. Hence, the cut type must be an arrow type. Now ctype then tells us that this arrow type must be either equal or strictly larger than the type of the created λ-abstraction. Thus, we can see that recursively reducing the application of the results of recursively applying hereditary substitution to t_1 and t_2 terminates based on our ordering. This explanation reveals that ctype is instrumental in the detection of newly created redexes and in proving properties of the hereditary substitution function.

The full normalization proof for STLC using hereditary substitution can be found in [10]. The following example gives some intuition of how the hereditary substitution function operates.

Example 11. *Consider the terms* $t \equiv \lambda f : b \to b.f$ *and* $t' \equiv (x\,(\lambda y : b.y))\,z$*, where z is a free variable of type b. Our goal is to compute* $[t/x]^{((b\to b)\to(b\to b))} t'$ *using the definition of the hereditary substitution function in Definition 13. First,*

$$[t/x]^{((b\to b)\to(b\to b))}(x\,(\lambda y : b.y)) = \lambda y : b.y,$$

because

$$\mathsf{ctype}_{((b\to b)\to(b\to b))}(x, x) = (b \to b) \to (b \to b),$$

$$[t/x]^{((b\to b)\to(b\to b))} x = t,$$

$$[t/x]^{((b\to b)\to(b\to b))}(\lambda y : b.y) = \lambda y : b.y,$$

and

$$[(\lambda y : b.y)/f]^{(b\to b)} f = \lambda y : b.y.$$

Now the previous facts give us that

$$[t/x]^{((b\to b)\to(b\to b))} t' = z,$$

because $[t/x]^{((b\to b)\to(b\to b))} z = z$, *and* $[z/y]^{b} y = z$.

5 Extending The Hereditary Substitution Function to the $\lambda\Delta$-Calculus

Since the $\lambda\Delta$-calculus is an extension of STLC, we might expect that the hereditary substitution function for the $\lambda\Delta$-calculus is also an extension of the hereditary substitution function for STLC. In this section we show that this extension is non-trivial by first considering the naive extension, and then discussing why it does not work. Following this, we give the final extension and prove it correct.

5.1 Problems with a Naive Extension

Lets consider the definition of the hereditary substitution function for STLC extended with two new cases. The first case for the Δ-abstraction whose definition parallels the definition for the λ-abstraction. The second is a new application case which handles newly created structural redexes and is defined following the same pattern as the case which handles β-redexes. We use the same termination metric we previously used.

Definition 12. *The naive hereditary substitution function is defined as follows:*

$$[t/x]^{A} x = t$$

$$[t/x]^{A} y = y$$

$$[t/x]^{A}(\lambda y : A'.t') = \lambda y : A'.([t/x]^{A} t')$$

$$[t/x]^{A}(\Delta y : A'.t') = \Delta y : A'.([t/x]^{A} t')$$

$$[t/x]^{A}(t_1\, t_2) = ([t/x]^{A} t_1)\,([t/x]^{A} t_2)$$
Where $([t/x]^{A} t_1)$ *is not a* λ*-abstraction or* Δ*-abstraction, or both* $([t/x]^{A} t_1)$ *and* t_1 *are* λ*-abstractions or* Δ*-abstractions.*

$$[t/x]^{A}(t_1\, t_2) = [s_2'/y]^{A''} s_1'$$
Where $([t/x]^{A} t_1) = \lambda y : A''.s_1'$ *for some* y, s_1' *and* A'', $[t/x]^{A} t_2 = s_2'$, *and* $\mathsf{ctype}_A(x, t_1) = A'' \to A'$.

$$[t/x]^{A}(t_1\, t_2) = \Delta z : \neg A'.[\lambda y : A'' \to A'.(z\,(y\, s_2))/y]^{\neg(A''\to A')} s$$
Where $([t/x]^{A} t_1) = \Delta y : \neg(A'' \to A').s$ *for some,* y s, *and* $A'' \to A'$, $([t/x]^{A} t_2) = s_2$ *for some* s_2, $\mathsf{ctype}_A(x, t_1) = A'' \to A'$, *and* z *is completely fresh.*

There is one glaring issue with this definition and it lies in the final case. We know from Lemma 8 and Lemma 15 that $\mathsf{ctype}_A(x, t_1) = A'' \to A'$ implies that $A \geq A'' \to A' < \neg(A'' \to A')$. Thus, this definition is not well founded! To fix this issue instead of naively following the structural reduction rule we immediately simultaneously hereditarily reduce all redexes created by replacing y with the linear λ-abstraction $\lambda y : A'' \to A'.(z\,(y\, s_2))$. To accomplish this we will define mutually with the hereditary substitution function a new function called the hereditary structural substitution function.

5.2 A Correct Extension of Hereditary Substitution

In order to reduce structural redexes in the definition of the hereditary substitution we will define by induction mutually with the hereditary substitution function a function called the hereditary structural substitution function. This function will use the notion of a multi-substitution. These are given by the following grammar:

$$\Theta ::= \cdot \mid \Theta, (y, z, t)$$

We denote the hereditary structural substitution function by $\langle\Theta\rangle^{A}_{A'}t'$ and hereditary substitution by $[t/x]^{A}t'$. The type of all the first projections of the elements of Θ is $\neg(A \to A')$ and the type of the second projections is $\neg A'$. Both functions are defined by mutual induction using the metric (A, f, t'), where $f \in \{0, 1\}$, in lexicographic combination with the ordering on types, the natural number ordering, and the strict subexpression on terms. The meta-variable f labels each function and is equal to 0 in the definition of the hereditary substitution function and is equal to 1 in the definition of the hereditary structural substitution function. Again, in the definitions of the hereditary substitution and hereditary structural substitution function it is assumed that all variables have been renamed as to prevent variable capture. The following is the final definition of the hereditary substitution function for the $\lambda\Delta$-calculus.

Definition 13. *The hereditary substitution function is defined as follows:*

$\langle\Theta\rangle^{A_1}_{A_2}x = \lambda y : A_1 \to A_2.(z\,(y\,t))$
Where $(x,z,t) \in \Theta$, for some z and t, and y is fresh in x, z, and t.

$\langle\Theta\rangle^{A_1}_{A_2}x = x$
Where $(x,z,t) \notin \Theta$ for any z or t.

$\langle\Theta\rangle^{A_1}_{A_2}(\lambda y : A.t) = \lambda y : A.\langle\Theta\rangle^{A_1}_{A_2}t$

$\langle\Theta\rangle^{A_1}_{A_2}(\Delta y : A.t) = \Delta y : A.\langle\Theta\rangle^{A_1}_{A_2}t$

$\langle\Theta\rangle^{A_1}_{A_2}(x\,t') = z\,[t/y]^{A_1}s$
Where $(x,z,t) \in \Theta$, $t' \equiv \lambda y : A_1.t''$, for some y and t'', and $\langle\Theta\rangle^{A_1}_{A_2}t'' = s$.

$\langle\Theta\rangle^{A_1}_{A_2}(x\,t') = z\,(\Delta z_2 : \neg A_2.s)$
Where $(x,z,t) \in \Theta$, $t' \equiv \Delta y : \neg(A_1 \to A_2).t''$, for some y and t'', and $\langle\Theta, (y, z_2, t)\rangle^{A_1}_{A_2}t'' = s$, for some fresh z_2.

$\langle\Theta\rangle^{A_1}_{A_2}(x\,t') = z\,s'$
Where $(x,z,t) \in \Theta$, t' is not an abstraction, and $\langle\Theta\rangle^{A_1}_{A_2}t' = s'$.

$\langle\Theta\rangle^{A_1}_{A_2}(t_1\,t_2) = s_1\,s_2$
Where t_1 is either not a variable, or it is both a variable and $(t_1, z', t') \notin \Theta$ for any t' and z', $\langle\Theta\rangle^{A_1}_{A_2}t_1 = s_1$, and $\langle\Theta\rangle^{A_1}_{A_2}t_2 = s_2$.

$[t/x]^{A}x = t$

$[t/x]^A y = y$

$[t/x]^A(\lambda y : A'.t') = \lambda y : A'.([t/x]^A t')$

$[t/x]^A(\Delta y : A'.t') = \Delta y : A'.([t/x]^A t')$

$[t/x]^A(t_1\, t_2) = ([t/x]^A t_1)\,([t/x]^A t_2)$
Where $([t/x]^A t_1)$ *is not a* λ*-abstraction or* Δ*-abstraction, or both* $([t/x]^A t_1)$ *and* t_1 *are* λ*-abstractions or* Δ*-abstractions.*

$[t/x]^A(t_1\, t_2) = [s_2'/y]^{A''} s_1'$
Where $([t/x]^A t_1) = \lambda y : A''.s_1'$ *for some* y, s_1' *and* A'', $[t/x]^A t_2 = s_2'$, *and* $\mathsf{ctype}_A(x, t_1) = A'' \to A'$.

$[t/x]^A(t_1\, t_2) = \Delta z : \neg A'.\langle (y, z, s_2)\rangle_{A'}^{A''} s$
Where $([t/x]^A t_1) = \Delta y : \neg(A'' \to A').s$ *for some* y s, *and* $A'' \to A'$, $([t/x]^A t_2) = s_2$ *for some* s_2, $\mathsf{ctype}_A(x, t_1) = A'' \to A'$, *and* z *is fresh.*

We can see in the final case of the hereditary substitution function that the cut type has decreased. Hence, this case is now well founded. Lets consider an example which illustrates how our new definition operates.

Example 14. *Consider the terms* $t \equiv \Delta f : \neg(b \to b).(f\,(\Delta f' : \neg(b \to b).(f'\,(\lambda z : b.z))))$ *and* $t' \equiv x\,u$, *where* u *is a free variable of type* b. *Again, our goal is to compute* $[t/x]^{(b\to b)} t'$ *using the definition of the hereditary substitution function in Definition 13. Now*

$$[t/x]^{(b\to b)}(x\,u) = \Delta z_1 : \neg b.(z_1\,(\Delta z_2 : \neg b.(z_2\,u))),$$

because

$$\mathsf{ctype}_{(b\to b)}(x, x) = (b \to b), \qquad [t/x]^{(b\to b)} x = t, \quad [t/x]^{(b\to b)} u = u,$$

and for some fresh variable z_1 *of type* $\neg b$

$$\begin{array}{l} \Delta z_1 : \neg b.\langle (f, z_1, u)\rangle_b^b (f\,(\Delta f' : \neg(b \to b).(f'\,(\lambda z : b.z)))) \quad = \\ \Delta z_1 : \neg b.(z_1\,(\Delta z_2 : \neg b.(z_2\,u))) \end{array}$$

where

$$\begin{array}{l} \langle (f, z_1, u)\rangle_b^b (f\,(\Delta f' : \neg(b \to b).(f'\,(\lambda z : b.z)))) \qquad = \\ z_1\,(\Delta z_2 : \neg b.\langle (f, z_1, u), (f', z_2, u)\rangle_b^b (f'\,(\lambda z : b.z))) \end{array}$$

because

$$(f, z_1, u) \in \langle (f, z_1, u)\rangle, \Delta f' : \neg(b \to b).(f'\,(\lambda z : b.z)) \equiv \Delta f' : \neg(b \to b).(f'\,(\lambda z : b.z)),$$

and for some fresh variable z_2 *of type* $\neg b$

$$\langle (f, z_1, u), (f', z_2, u)\rangle_b^b (f'\,(\lambda z : b.z)) \quad = \quad z_2\,u$$

because

$$(f', z_2, u) \in \langle (f, z_1, u), (f', z_2, u)\rangle, \qquad \lambda z : b.z \equiv \lambda z : b.z, \quad \langle (f, z_1, u)\rangle_b^b z = z$$

In the next section we prove the definition of the hereditary substitution function correct.

5.3 Properties of the Hereditary Substitution Function

There are two main ways the hereditary substitution function is used. It either replaces capture-avoiding substitution in ones' type theory or it is used in some other way. For example, in Canonical LF hereditary substitution replaces capture-avoiding substitution [18, 3]. However, in [2] it is used only as a normalization function. No matter how it is used there are three correctness results which must be proven. These are totality, type preservation, and normality preservation. There is an additional correctness property we feel one must prove when hereditary substitution is used as a normalization function. This property is called soundness with respect to reduction. It shows that hereditary substitution does nothing more than what capture-avoiding substitution and β-reduction can do.

We introduce some notation to make working with multi-substitutions a bit easier. The sets of all first, second, and third projections of the triples in Θ are denoted Θ^1, Θ^2, and Θ^3 respectively. We denote the assumption of all elements of Θ^i having the type T as $\Theta^i : T$. This latter notation is used in typing contexts to indicate the addition of all the variables in Θ^j for $j \in \{1,2\}$ to the context with the specified type. We denote this as $\Gamma, \Theta^j : T, \Gamma'$ for some contexts Γ and Γ'. The notation $\Gamma \vdash \Theta^3 : T$ is defined as for all $t \in \Theta^3$ the typing judgment $\Gamma \vdash t : T$ holds. Finally, we denote terms in Θ^3 being normal as $\mathsf{norm}(\Theta^3)$.

All of the following properties will depend on a few more properties of the ctype function. They are listed in the following lemma.

Lemma 15 (Properties of ctype Continued)**.**

i. If $\Gamma, x : T, \Gamma' \vdash t_1\, t_2 : T'$, $\Gamma \vdash t : T$, $[t/x]^T t_1 = \lambda y : T_1.t'$, and t_1 is not a λ-abstraction, then t_1 is in head normal form and there exists a type A such that $\mathsf{ctype}_T(x, t_1) = A$.

ii. If $\Gamma, x : T, \Gamma' \vdash t_1\, t_2 : T'$, $\Gamma \vdash t : T$, $[t/x]^T t_1 = \Delta y : \neg(T'' \to T').t'$, and t_1 is not a Δ-abstraction, then there exists a type A such that $\mathsf{ctype}_T(x, t_1) = A$.

Proof. Both parts can be shown by induction on the structure of $t_1\, t_2$. See Appendix A.2. □

These are all similar to the properties in Lemma 10. The first two properties of the hereditary substitution function are totality and type preservation. The latter is similar to substitution for typing using the hereditary substitution function.

Lemma 16 (Totality and Type Preservation)**.**

i. If $\Gamma \vdash \Theta^3 : A$ and $\Gamma, \Theta^1 : \neg(A \to A') \vdash t' : B$, then there exists a term s such that $\langle\Theta\rangle^A_{A'} t' = s$ and $\Gamma, \Theta^2 : \neg A' \vdash s : B$.

ii. If $\Gamma \vdash t : A$ and $\Gamma, x : A, \Gamma' \vdash t' : B$, then there exists a term s such that $[t/x]^A t' = s$ and $\Gamma, \Gamma' \vdash s : B$.

Proof. This can be shown by mutual induction using the lexicographic combination (A, f, t') of our ordering on types, the natural number ordering where $f \in \{0,1\}$, and the strict subexpression ordering on terms. See Appendix A.3. □

The next property shows that the hereditary substitution function is normality preserving. That is, if the input to the hereditary substitution function is normal then so is the output. This is crucial for the normalization argument. The proof of normality preservation depends on the following auxiliary result.

Lemma 17. *For any Θ, A and A', if $n_1\, n_2$ is normal then $\mathsf{head}(\langle\Theta\rangle^A_{A'}(n_1\, n_2))$ is a variable.*

Proof. This proof is by induction on the form of $n_1\, n_2$. See Appendix A.4. □

Lemma 18 (Normality Preservation)**.**

i. *If* $\mathsf{norm}(\Theta^3)$, $\Gamma \vdash \Theta^3 : A$ *and* $\Gamma, \Theta^1 : \neg(A \to A') \vdash n' : B$, *then there exists a normal form* m *such that* $\langle\Theta\rangle^A_{A'} n' = m$.

ii. *If* $\Gamma \vdash n : A$ *and* $\Gamma, x : A, \Gamma' \vdash n' : B$ *then there exists a term* m *such that* $[n/x]^A n' = m$.

Proof. This can be shown by mutual induction using the lexicographic combination (A, f, t') of our ordering on types, the natural number ordering where $f \in \{0,1\}$, and the strict subexpression ordering on terms. See Appendix A.5. □

The final correctness property of the hereditary substitution function is soundness with respect to reduction. We need one last piece of notation. Suppose $\Theta = (x_1, z_1, t_1), \ldots, (x_i, z_i, t_i)$ for some natural number i. Then $\langle\Theta\rangle^{\uparrow^A_{A'}} t' =^{def} [\lambda y : A \to A'.(z_i\,(y\,t_i))/x_i](\cdots([\lambda y : A \to A'.(z_1\,(y\,t_1))/x_1]t_1)\cdots)$.

Lemma 19 (Soundness with Respect to Reduction)**.**

i. *If* $\Gamma \vdash \Theta^3 : A$ *and* $\Gamma, \Theta^1 : \neg(A \to A') \vdash t' : B$, *then* $\langle\Theta\rangle^{\uparrow^A_{A'}} t' \rightsquigarrow^* \langle\Theta\rangle^A_{A'} t'$.

ii. *If* $\Gamma \vdash t : A$ *and* $\Gamma, x : A, \Gamma' \vdash t' : B$ *then* $[t/x]t' \rightsquigarrow^* [t/x]^A t'$.

Proof. This can be shown by mutual induction using the lexicographic combination (A, f, t') of our ordering on types, the natural number ordering where $f \in \{0,1\}$, and the strict subexpression ordering on terms. See Appendix A.6. □

Using these properties it is now possible to conclude normalization for the $\lambda\Delta$-calculus.

6 Concluding Normalization

We now define the interpretation $[\![T]\!]_\Gamma$ of types T in typing context Γ. This is in fact the same interpretation of types that was used to show normalization using hereditary substitution of Stratified System F in [8].

Definition 20. *The interpretation of types* $[\![T]\!]_\Gamma$ *is defined by:*

$$n \in [\![T]\!]_\Gamma \iff \Gamma \vdash n : T$$

We extend this definition to non-normal terms t *in the following way:*

$$t \in [\![T]\!]_\Gamma \iff \exists n.t \rightsquigarrow^! n \in [\![T]\!]_\Gamma$$

Type soundness depends on the following lemma. It shows that the interpretation of types is closed under hereditary substitution.

Lemma 21 (Hereditary Substitution for the Interpretation of Types)**.** *If* $n \in [\![T]\!]_\Gamma$ *and* $n' \in [\![T']\!]_{\Gamma,x:T,\Gamma'}$, *then* $[n/x]^T n' \in [\![T']\!]_{\Gamma,\Gamma'}$.

Proof. We know by Lemma 16 that there exists a term s such that $[n/x]^T n' = s$ and $\Gamma, \Gamma' \vdash s : T'$, and by Lemma 18 s is normal. Therefore, $s \in [\![T']\!]_{\Gamma,\Gamma'}$. □

Using the previous lemma and the properties of the hereditary substitution function we can now prove type soundness.

Theorem 22 (Type Soundness)**.** *If* $\Gamma \vdash t : T$ *then* $t \in [\![T]\!]_\Gamma$.

The only hard case in the proof of the type soundness theorem is the case for applications. Using the previous lemma and the properties of the hereditary substitution function, however, it goes through with ease. Consider the application case of the proof of type soundness. Note that the proof is by induction on the assumed typing derivation (See Appendix A.7).

$$\frac{\Gamma \vdash t_2 : A \quad \Gamma \vdash t_1 : A \to B}{\Gamma \vdash t_1\, t_2 : B}\ \textsc{App}$$

By the induction hypothesis we know $t_1 \in [\![A \to B]\!]_\Gamma$ and $t_2 \in [\![A]\!]_\Gamma$. So by the definition of the interpretation of types we know there exists normal forms n_1 and n_2 such that $t_1 \rightsquigarrow^* n_1 \in [\![A \to B]\!]_\Gamma$ and $t_2 \rightsquigarrow^* n_2 \in [\![A]\!]_\Gamma$. Assume y is a fresh variable in n_1 and n_2 of type A. Then by hereditary substitution for the interpretation of types (Lemma 21) $[n_1/y]^A(y\,n_2) \in [\![B]\!]_\Gamma$. It suffices to show that $t_1\,t_2 \rightsquigarrow^* [n_1/y]^A(y\,n_2)$. This is an easy consequence of soundness with respect to reduction (Lemma 19), that is, $t_1\,t_2 \rightsquigarrow^* n_1\,n_2 = [n_1/y](y\,n_2)$ and by soundness with respect to reduction $[n_1/y](y\,n_2) \rightsquigarrow^* [n_1/y]^A(y\,n_2)$. Therefore, $t_1\,t_2 \in [\![B]\!]_\Gamma$.

Finally, we conclude normalization for the $\lambda\Delta$-calculus using hereditary substitution.

Corollary 23 (Normalization)**.** *If $\Gamma \vdash t : T$ then there exists a term n such that $t \rightsquigarrow^! n$.*

7 Related Work

We first compare the proof method normalization using hereditary substitution with other known proof methods. The $\lambda\Delta$-calculus could have been proven weakly and strongly normalizing by translation to $\lambda\mu$-calculus. It is true that this is not as complicated as the proof method here, but a proof by translation does not yield a direct proof.

A direct proof of weak and strong normalization could have been given using the Tait-Girard reducibility method. However, we claim that the proof method used here is less complicated. The statement of the type soundness theorem is qualitatively less complex due to the fact that there is no need to universally quantify over the set of well-formed substitutions. We are able to prove type soundness on open terms directly. Additionally, the formalization of normalization using hereditary substitution does not require recursive types to define the semantics of types which are required when formalizing a proof using reducibility.

R. David and K. Nour give a short proof of normalization of the $\lambda\Delta$-calculus in [7]. There they use a rather complicated lexicographic combination to give a completely arithmetical proof of strong normalization. While they show strong normalization their proof method is comparable to using hereditary substitution. As we mentioned in the introduction hereditary substitution is the constructive content of normalization proofs using the lexicographic combination of an ordering on types and the strict subexpression ordering on terms. It is currently unknown if hereditary substitution can be extended to show strong normalization, but we conjecture that the constructive content of the proof of Lemma 3..6 in David and Nour's work would yield a hereditary substitution like function. Furthermore, for simply typed theories we believe it is enough to show weak normalization and never need to show strong normalization. It is well-known due to the work of G. Barthe et al. in [6] that for the entire left hand side of the λ-cube weak normalization implies strong normalization. We conjecture that this result would extend to the left hand side of the classical λ-cube given in [5]. Thus, showing normalization using hereditary substitution is less complicated than the work of David and Nour's.

Similar to the work of David and Nour is the work of F. Joachimski and R. Matthes. In [11] they prove weak and strong normalization of various simply typed theories. The proof method used is induction on various lexicographic combinations similar to hereditary substitution. After proving weak normalization of each type theory they extract the constructive content of the proof yielding a normalization function which depends on a substitution function similar to the hereditary substitution function. In contrast once hereditary substitution is defined for a type theory we can easily define a normalization function. Note that the following function is the computational content of the type-soundness theorem (Theorem 22).

Definition 24. *We define a normalization function for the $\lambda\Delta$-calculus using hereditary substitution as follows:*

$$\mathsf{norm}\, x = x$$

$$\mathsf{norm}\,(\lambda x : A.t) = \lambda x : A.(\mathsf{norm}\, t)$$

$$\mathsf{norm}\,(\Delta x : A.t) = \Delta x : A.(\mathsf{norm}\, t)$$

$$\mathsf{norm}\,(t_1\, t_2) = [n_1/r]^A(r\, n_2)$$
Where $\mathsf{norm}\, t_1 = n_1$, $\mathsf{norm}\, t_2 = n_2$, *$A$ is the type of t_1, and r is fresh in t_1 and t_2.*

This function is similar to the normalization functions in Joachimski and Matthes' work. We could use the above normalization function to decide $\beta\eta$-equality for the $\lambda\Delta$-calculus. Indeed this one of the main application of hereditary substitution.

A. Abel in 2006 shows how to implement a normalizer using sized heterogeneous types which is a function similar to the hereditary substitution function in [1]. He then uses hereditary substitution to prove normalization of the type level of a type theory with higher-order subtyping in [2]. This results in a purely syntactic metatheory. C. Keller and T. Altenkirch recently implemented hereditary substitution as a normalization function for the simply typed λ-calculus in Agda [12]. Their results show that hereditary substitution can be used to decide $\beta\eta$-equality. They found hereditary substitution to be convenient to use in a total type theory, because it can be implemented without a termination proof. This is because the hereditary-substitution function can be recognized as structurally recursive, and hence accepted directly by Agda's termination checker.

One point which sets the current work apart from all of the related work just considered is that they were all concerned with intuitionistic type theories. Here we apply hereditary substitution on a classical type theory. To our knowledge this is the first time this has been done.

8 Conclusion

We briefly gave an overview of the hereditary substitution proof method for showing normalization of typed λ-calculi and showed how to extend and apply it to the $\lambda\Delta$-calculus. In Section 5.2 we defined the hereditary substitution function for the $\lambda\Delta$-calculus which involved a new function called the structural hereditary substitution function. Then we proved the main properties of the hereditary substitution function in Section 5.3. Lastly, we concluded normalization in Section 6.

Future work. The authors conjecture that the current work may extend to yield a direct proof of normalization of the $\lambda\mu$-calculus using hereditary substitution.

References

[1] Andreas Abel (2006): *Implementing a normalizer using sized heterogeneous types*. In: *In Workshop on Mathematically Structured Functional Programming, MSFP*, doi:10.1017/S0956796809007266.

[2] Andreas Abel & Dulma Rodriguez (2008): *Syntactic Metatheory of Higher-Order Subtyping*. In: *Proceedings of the 22nd international workshop on Computer Science Logic*, CSL '08, Springer-Verlag, Berlin, Heidelberg, pp. 446–460, doi:10.1007/978-3-540-87531-4_32.

[3] Robin Adams (2004): *A Modular Hierarchy of Logical Frameworks*. Ph.D. thesis.

[4] R.M. Amadio & P.L. Curien (1998): *Domains and lambda-calculi*. Cambridge tracts in theoretical computer science, Cambridge University Press, doi:10.1017/CBO9780511983504.

[5] G. Barthe, J. Hatcliff & M. Heine Sørensen (1997): *A notion of classical pure type system (preliminary version)*. *Electronic Notes in Theoretical Computer Science* 6, pp. 4–59, doi:10.1016/S1571-0661(05)80170-7.

[6] Gilles Barthe, John Hatcliff & Morten Heine Srensen (2001): *Weak normalization implies strong normalization in a class of non-dependent pure type systems*. *Theoretical Computer Science* 269(1-2), pp. 317 – 361, doi:10.1016/S0304-3975(01)00012-3.

[7] Rene David & Karim Nour (2003): *A short proof of the strong normalization of the simply typed lambdamu-calculus*. *SCHEDAE INFORMATICAE* 12, pp. 27–33.

[8] Harley Eades & Aaron Stump (2010): *Hereditary Substitution for Stratified System F*. In: *Proof-Search in Type Theories (PSTT)*.

[9] Jean-Yves Girard, Yves Lafont & Paul Taylor (1989): *Proofs and Types (Cambridge Tracts in Theoretical Computer Science)*. Cambridge University Press.

[10] Harley Eades II & Aaron Stump (2011): *Using the Hereditary Substitution Function in Normalization Proofs*. Available at `http://metatheorem.org/wp-content/papers/qual_companion_report.pdf`.

[11] Felix Joachimski & Ralph Matthes (1999): *Short Proofs of Normalization for the simply-typed lambda-calculus, permutative conversions and Gödel's T*.

[12] Chantal Keller & Thorsten Altenkirch (2010): *Hereditary substitutions for simple types, formalized*. In: *Proceedings of the third ACM SIGPLAN workshop on Mathematically structured functional programming*, MSFP '10, ACM, New York, NY, USA, pp. 3–10, doi:10.1145/1863597.1863601.

[13] D. Leivant (1991): *Finitely stratified polymorphism*. *Inf. Comput.* 93(1), pp. 93–113, doi:10.1016/0890-5401(91)90053-5.

[14] Jean-Jacques Lévy (1976): *An algebraic interpretation of the K-calculus; and an application of a labelled -calculus*. *Theoretical Computer Science* 2(1), pp. 97 – 114, doi:10.1016/0304-3975(76)90009-8.

[15] Michel Parigot (1992): *Lambda-Mu-Calculus: An algorithmic interpretation of classical natural deduction*. In Andrei Voronkov, editor: *Logic Programming and Automated Reasoning*, *Lecture Notes in Computer Science* 624, Springer Berlin / Heidelberg, pp. 190–201, doi:10.1007/BFb0013061.

[16] Michel Parigot (1997): *Proofs of Strong Normalization for Second Order Classical Natural Deduction*. *Journal of Symbolic Logic* 62(4), pp. 1461–1479, doi:10.2307/2275652.

[17] Jakob Rehof & Morten Heine Sørensen (1994): *The LambdaDelta-calculus*. In: *Proceedings of the International Conference on Theoretical Aspects of Computer Software*, TACS '94, Springer-Verlag, London, UK, pp. 516–542, doi:10.1007/3-540-57887-0_113.

[18] Kevin Watkins, Iliano Cervesato, Frank Pfenning & David Walker (2004): *A Concurrent Logical Framework: The Propositional Fragment*. In Stefano Berardi, Mario Coppo & Ferruccio Damiani, editors: *Types for Proofs and Programs*, *Lecture Notes in Computer Science* 3085, Springer Berlin / Heidelberg, pp. 355–377, doi:10.1007/978-3-540-24849-1_23.

A Proofs

A.1 Proof of Properties of ctype_T

We prove part one first. This is a proof by induction on the structure of t.

Case. Suppose $t \equiv x$. Then $\mathsf{ctype}_T(x,x) = T$. Clearly, $\mathsf{head}(x) = x$ and T is a subexpression of itself.

Case. Suppose $t \equiv t_1\, t_2$. Then $\mathsf{ctype}_T(x, t_1\, t_2) = T''$ when $\mathsf{ctype}_T(x,t_1) = T' \to T''$. Now $t > t_1$ so by the induction hypothesis $\mathsf{head}(t_1) = x$ and $T' \to T''$ is a subexpression of T. Therefore, $\mathsf{head}(t_1\, t_2) = x$ and certainly T'' is a subexpression of T.

We now prove part two. This is also a proof by induction on the structure of t.

Case. Suppose $t \equiv x$. Then $\mathsf{ctype}_T(x,x) = T$. Clearly, $T \equiv T$.

Case. Suppose $t \equiv t_1\, t_2$. Then $\mathsf{ctype}_T(x, t_1\, t_2) = T_2$ when $\mathsf{ctype}_T(x,t_1) = T_1 \to T_2$. By inversion on the assumed typing derivation we know there exists type T'' such that $\Gamma, x : T, \Gamma' \vdash t_1 : T'' \to T'$. Now $t > t_1$ so by the induction hypothesis $T_1 \to T_2 \equiv T'' \to T'$. Therefore, $T_1 \equiv T''$ and $T_2 \equiv T'$.

A.2 Proof of Properties of ctype_T Continued

We prove part one first. This is a proof by induction on the structure of $t_1\, t_2$.

The only possibilities for the form of t_1 is x or $s_1\, s_2$. All other forms would not result in $[t/x]^T t_1$ being a λ-abstraction and t_1 not. If $t_1 \equiv x$ then there exist a type T'' such that $T \equiv T'' \to T'$ and $\mathsf{ctype}_T(x, x\, t_2) = T'$ when $\mathsf{ctype}_T(x,x) = T \equiv T'' \to T'$ in this case. We know T'' to exist by inversion on $\Gamma, x : T, \Gamma' \vdash t_1\, t_2 : T'$.

Now suppose $t_1 \equiv s_1\, s_2$. Now knowing t_1 to not a λ-abstraction implies that s_1 is also not a λ-abstraction or $[t/x]^T t_1$ would be an application instead of a λ-abstraction. So it must be the case that $[t/x]^T s_1$ is a λ-abstraction and s_1 is not. Since $s_1 < t_1$ we can apply the induction hypothesis to obtain there exists a type A such that $\mathsf{ctype}_T(x, s_1) = A$. Now by inversion on $\Gamma, x : T, \Gamma' \vdash t_1\, t_2 : T'$ we know there exists a type T'' such that $\Gamma, x : T, \Gamma' \vdash t_1 : T'' \to T'$. We know $t_1 \equiv s_1\, s_2$ so by inversion on $\Gamma, x : T, \Gamma' \vdash t_1 : T'' \to T'$ we know there exists a type A'' such that $\Gamma, x : T, \Gamma' \vdash s_1 : A'' \to (T'' \to T')$. By part two of Lemma 8 we know $A \equiv A'' \to (T'' \to T')$ and $\mathsf{ctype}_T(x, t_1) = \mathsf{ctype}_T(x, s_1\, s_2) = T'' \to T'$ when $\mathsf{ctype}_T(x, s_1) = A'' \to (T'' \to A')$, because we know $\mathsf{ctype}_T(x, s_1) = A$.

The proof of part two is similar to the proof of part one.

A.3 Proof of Totality and Type Preservation

This is a mutually inductive proof using the lexicographic combination (A, f, t') of our ordering on types, the natural number ordering where $f \in \{0, 1\}$, and the strict subexpression ordering on terms. We first prove part one and then part two. In both parts we case split on t'.

Part One.

Case. Suppose t' is a variable x. Then either there exists a term a such that $(x, z, a) \in \Theta$ or not. Suppose so. Then $\langle \Theta \rangle^A_{A'} x = \lambda y : A \to A'.(z\,(y\,a))$ where y is fresh in x, z and a. Now suppose there does

not exist any term a or z such that $(x,z,a) \in \Theta$. Then $\langle\Theta\rangle^A_{A'}x = x$. Typing clearly holds, because if $(x,z,a) \in \Theta$ then $B \equiv \neg(A \to A')$ and we know $\Gamma, \Theta^2 : \neg A' \vdash \lambda y : A \to A'.(z\,(y\,a)) : B$ or $x \notin \Theta^1$ then it must be the case that $x : B \in \Gamma$, hence, by assumption and weakening for typing $\Gamma, \Theta^2 : \neg A' \vdash x : B$.

Case. It must be the case that $B \equiv B_1 \to B_2$ for some types B_1 and B_2. Suppose $t' \equiv \lambda y : B_1.t'_1$. Then $\langle\Theta\rangle^A_{A'}t' = \langle\Theta\rangle^A_{A'}(\lambda y : B_1.t'_1) = \lambda y : B_1.\langle\Theta\rangle^A_{A'}t'_1$. Now sense $(A,1,t') > (A,1,t'_1)$ we may apply the induction hypothesis to obtain that there exists a term s such that $\langle\Theta\rangle^A_{A'}t'_1 = s$, and $\Gamma, \Theta^2 : \neg A', y : B_1 \vdash s : B_2$. Thus, by definition and the typing rule for λ-abstractions we obtain $\langle\Theta\rangle^A_{A'}t' = \lambda y : B_1.s$ and $\Gamma, \Theta^2 : \neg A' \vdash \lambda y : B_1.s : B_1 \to B_2$.

Case. Suppose $t' \equiv \Delta y : \neg B.t'_1$. Similar to the previous case.

Case. Suppose $t' \equiv t'_1\,t'_2$. We have two cases to consider.

Case. Suppose $t'_1 \equiv x$ for some variable x. In each case $B \equiv \bot$.

Case. Suppose $t'_2 \equiv \lambda y : A.t''_2$, for some y and t''_2, $(x,z,t) \in \Theta$. Since $(A,1,t') > (A,1,t''_2)$ and the typing assumptions hold by inversion we can apply the induction hypothesis to obtain $\langle\Theta\rangle^A_{A'}t''_2 = s$ for some term s and $\Gamma, \Theta^2 : \neg A', y : A \vdash s : A'$. Furthermore, sense $(A,1,t') > (A,0,s)$, the previous typing condition and the typing assumptions we also know from the induction hypothesis that $[t/y]^A s = s'$ for some term s' and $\Gamma, \Theta^2 : \neg A' \vdash s' : A'$. Finally, by definition we know $\langle\Theta\rangle^A_{A'}t' = z\,([t/y]^A s) = z\,s'$ and by using the application typing rule that $\Gamma, \Theta^2 : \neg A' \vdash z\,s' : B$.

Case. Suppose $t'_2 \equiv \Delta y : \neg(A \to A').t''_2$, for some y and t''_2, $(x,z,t) \in \Theta$. Since $(A,1,t') > (A,1,t''_2)$ we know from the induction hypothesis that $\langle\Theta,(y,z_2,t)\rangle^A_{A'}t''_2 = s$ for some fresh variable z and term s, and $\Gamma, \Theta^2 : \neg A', z_2 : \neg A' \vdash s : \bot$. Finally, $\langle\Theta\rangle^A_{A'}t' = z\,(\Delta z_2 : \neg A'.s)$ by definition, and by using the application typing rule $\Gamma, \Theta^2 : \neg A' \vdash z\,(\Delta z_2 : \neg A'.s) : B$.

Case. Suppose t'_2 is not an abstraction, and $(x,z,t) \in \Theta$. Since $(A,1,t') > (A,1,t'_2)$ we know from the induction hypothesis that $\langle\Theta\rangle^A_{A'}t'_2 = s$ for some term s and $\Gamma, \Theta^2 : \neg A' \vdash s : A \to A'$. Finally, $\langle\Theta\rangle^A_{A'}t' = z\,s$ by definition, and by using the application typing rule $\Gamma, \Theta^2 : \neg A' \vdash z\,s : B$.

Case. Suppose $(x,z,t'') \notin \Theta$ for any term t'' and z. Since $(A,1,t') > (A,1,t'_2)$ we know from the induction hypothesis that $\langle\Theta\rangle^A_{A'}t'_2 = s$ for some term s and $\Gamma, \Theta^2 : \neg A' \vdash s : A \to A'$. Finally, $\langle\Theta\rangle^A_{A'}t' = x\,s$ by definition, and by using the application typing rule $\Gamma, \Theta^2 : \neg A' \vdash x\,s : B$.

Case. Suppose t'_1 is not a variable. This case follows easily from the induction hypothesis.

Part two.

Case. Suppose t' is either x or a variable y distinct from x. Trivial in both cases.

Case. Suppose $t' \equiv \lambda y : A_1.t'_1$. By inversion we know there exists a type A_2 such that $\Gamma, x : A, \Gamma', y : A_1 \vdash t'_1 : A_2$. We also know that t'_1 is a strict subexpression of t', hence we can apply the second part of the induction hypothesis to obtain $[t/x]^A t'_1 = s_1$ and $\Gamma, \Gamma', y : A_1 \vdash s_1 : A_2$ for some term s_1. By the definition of the hereditary substitution function

$$\begin{aligned} [t/x]^A t' &= \lambda y : A_1.[t/x]^A t'_1 \\ &= \lambda y : A_1.s_1. \end{aligned}$$

It suffices to show that $\Gamma, \Gamma' \vdash \lambda y : A_1.s_1 : A_1 \to A_2$. By simply applying the typing rule LAM using $\Gamma, \Gamma', y : A_1 \vdash s_1 : A_2$ we obtain $\Gamma, \Gamma' \vdash \lambda y : A_1.s_1 : A_1 \to A_2$.

Case. Suppose $t' \equiv \Delta y : \neg B.t'_1$. Similar to the previous case.

Case. Suppose $t' \equiv t'_1\, t'_2$. By inversion we know $\Gamma, x : A, \Gamma' \vdash t'_1 : B' \to B$ and $\Gamma, x : A, \Gamma' \vdash t'_2 : B'$ for some type B'. Clearly, t'_1 and t'_2 are strict subexpressions of t'. Thus, by the second part of the induction hypothesis there exists terms s_1 and s_2 such that $[t/x]^A t'_1 = s_1$ and $[t/x]^A t'_2 = s_2$, and $\Gamma, \Gamma' \vdash s_1 : B' \to B'$ and $\Gamma, \Gamma' \vdash s_2 : B'$. We case split on whether or not s_1 is a λ-abstraction or a Δ-abstraction and t'_1 is not, or s_1 and t'_1 are both a λ-abstraction or a Δ-abstraction. We only consider the non-trivial cases when $s_1 \equiv \lambda y : B'.s'_1$ and t'_1 is not a λ-abstraction, and $s_1 \equiv \Delta y : \neg(B' \to B).s'_1$ and t'_1 is not a Δ-abstraction. Consider the former.

Now by Lemma 8 it is the case that there exists a B'' such that $\mathsf{ctype}_A(x, t'_1) = B''$, $B'' \equiv B' \to B$, and B is a subexpression of A, hence $A > B'$. By the definition of the hereditary substitution function $[t/x]^A(t'_1\, t'_2) = [s_2/y]^{B'} s'_1$. Therefore, by the induction hypothesis there exists a term s such that $[s_2/y]^A s'_1 = s$ and $\Gamma, \Gamma' \vdash s : B$.

At this point consider when $s_1 \equiv \Delta y : \neg(B' \to B).s'_1$ and t'_1 is not a Δ-abstraction. Again, by Lemma 8 it is the case that there exists a B'' such that $\mathsf{ctype}_A(x, t'_1) = B''$, $B'' \equiv B' \to B$ and $B' \to B$ is a subexpression of A. Hence, $A > B'$. Let r be a fresh variable of type $\neg B$. Then by the induction hypothesis, there exists a term s'', such that, $\langle (y, r, s_2) \rangle_B^{B'} s'_1 = s''$ and $\Gamma, r : \neg B \vdash s'' : \bot$. Therefore, $[t/x]^A(t'_1\, t'_2) = \Delta r : \neg B.\langle (y, r, s_2) \rangle_B^{B'} s'_1 = \Delta r : \neg B.s''$, and by the Δ-abstraction typing rule $\Gamma \vdash \Delta r : \neg B.s'' : B$.

A.4 Proof of Lemma 17

This is a proof by induction on the form of $n_1\, n_2$. In every case where n_1 is a variable and $(n_1, z, t) \in \Theta$ for some term t and variable z, we know by definition that $\langle \Theta \rangle_{A'}^A (n_1\, n_2) = z\, t_2$ for some variable z and term t_2. In the case where n_1 is a variable and $(n_1, z, t) \notin \Theta$ for some term t and variable z, we know by definition that $\langle \Theta \rangle_{A'}^A (n_1\, n_2) = (\langle \Theta \rangle_{A'}^A n_1)\, (\langle \Theta \rangle_{A'}^A n_2)$. Now by hypothesis and definition $\langle \Theta \rangle_{A'}^A n_1 = n_1$. Thus, $(\langle \Theta \rangle_{A'}^A n_1)\, (\langle \Theta \rangle_{A'}^A n_2) = n_1\, (\langle \Theta \rangle_{A'}^A n_2)$ and we know n_1 is a variable. The final case is when n_1 is not a variable. Then it must be the case that n_1 is a normal application. So by the induction hypothesis $\mathsf{head}(\langle \Theta \rangle_{A'}^A n_1)$ is a variable. Therefore, $\mathsf{head}(\langle \Theta \rangle_{A'}^A (n_1\, n_2)) = \mathsf{head}((\langle \Theta \rangle_{A'}^A n_1)\, (\langle \Theta \rangle_{A'}^A n_2)) = \mathsf{head}(\langle \Theta \rangle_{A'}^A n_1)$ is a variable.

A.5 Proof of Normality Preservation

This is a mutually inductive proof using the lexicographic combination (A, f, n') of our ordering on types, the natural number ordering where $f \in \{0, 1\}$, and the strict subexpression ordering on terms. We first prove part one and then part two. In both parts we case split on n'.

Part One.

Case. Suppose n' is a variable x. Then either there exists a normal form m and variable z, such that, $(x, z, m) \in \Theta$ or not. Suppose so. Then $\langle \Theta \rangle_{A'}^A x = \lambda y : A \to A'.(z\,(y\,m))$ where y is fresh in x, z and m. Clearly, $\lambda y : A \to A'.(z\,(y\,m))$ is normal. Now suppose there does not exist any term m or z such that $(x, z, m) \in \Theta$. Then $\langle \Theta \rangle_{A'}^A x = x$ which is clearly normal.

Case. Suppose $n' \equiv \lambda y : B_1.n'_1$. Then $\langle\Theta\rangle^A_{A'}n' = \langle\Theta\rangle^A_{A'}(\lambda y : B_1.n'_1) = \lambda y : B_1.\langle\Theta\rangle^A_{A'}n'_1$. Now sense $(A,1,n') > (A,1,n'_1)$ we may apply the induction hypothesis to obtain that there exists a term m such that $\langle\Theta\rangle^A_{A'}n'_1 = m$. Thus, by definition we obtain $\langle\Theta\rangle^A_{A'}n' = \lambda y : B_1.m$.

Case. Suppose $n' \equiv \Delta y : \neg B.n'_1$. Similar to the previous case.

Case. Suppose $n' \equiv n'_1\, n'_2$. We have two cases to consider.

Case. Suppose $n'_1 \equiv x$ for some variable x.

Case. Suppose $n'_2 \equiv \lambda y : A.n''_2$, for some y and n''_2, $(x,z,n) \in \Theta$. Since $(A,1,n') > (A,1,n''_2)$ and the typing assumptions hold by inversion we can apply the induction hypothesis to obtain $\langle\Theta\rangle^A_{A'}n''_2 = m$ for some term m. We know from Lemma 18 that $\Gamma, \Theta^2 : \neg A', y : A \vdash m : A'$. Furthermore, sense $(A,1,n') > (A,0,m)$, the previous typing condition and the typing assumptions we also know from the induction hypothesis that $[t/y]^A m = m'$ for some term m'. Finally, by definition we know $\langle\Theta\rangle^A_{A'}n' = z([n/y]^A m) = z\,m'$. It is easy to see that $z\,m'$ is normal.

Case. Suppose $n'_2 \equiv \Delta y : \neg(A \to A').n''_2$, for some y and n''_2, $(x,z,n) \in \Theta$. Since $(A,1,n') > (A,1,n''_2)$ we know from the induction hypothesis that $\langle\Theta,(y,z_2,n)\rangle^A_{A'}n''_2 = m$ for some normal form m, and $\Gamma, \Theta^2 : \neg A', z_2 : \neg A' \vdash m : \bot$. Finally, $\langle\Theta\rangle^A_{A'}n' = z(\Delta z_2 : \neg A'.m)$ by definition.

Case. Suppose n'_2 is not an abstraction, and $(x,z,n) \in \Theta$. Since $(A,1,n') > (A,1,n'_2)$ we know from the induction hypothesis that $\langle\Theta\rangle^A_{A'}n'_2 = m$ for some normal form m. Finally, $\langle\Theta\rangle^A_{A'}n' = z\,m$ by definition.

Case. Suppose $(x,z,n'') \notin \Theta$ for any term n'' and z. Since $(A,1,n') > (A,1,n'_2)$ we know from the induction hypothesis that $\langle\Theta\rangle^A_{A'}n'_2 = m$ for some term m. Finally, $\langle\Theta\rangle^A_{A'}n' = x\,m$ by definition.

Case. Suppose n'_1 is not a variable. This case follows easily from the induction hypothesis and Lemma 17.

Part two.

Case. Suppose n' is either x or a variable y distinct from x. Trivial in both cases.

Case. Suppose $n' \equiv \lambda y : B_1.n'_1$. We also know that n'_1 is a strict subexpression of n', hence we can apply the second part of the induction hypothesis to obtain $[n/x]^A n'_1 = m_1$ for some normal form m_1. By the definition of the hereditary substitution function

$$\begin{array}{rcl} [n/x]^A n' & = & \lambda y : A_1.[n/x]^A n'_1 \\ & = & \lambda y : B_1.m_1. \end{array}$$

Clearly, $\lambda y : B_1.m_1$ is normal.

Case. Suppose $n' \equiv \Delta y : \neg B.n'_1$. Similar to the previous case.

Case. Suppose $t' \equiv t'_1\, t'_2$. Clearly, n'_1 and n'_2 are strict subexpressions of n'. Thus, by the induction hypothesis there exists normal forms n_1 and n_2 such that $[n/x]^A n'_1 = m_1$ and $[n/x]^A n'_2 = m_2$. We case split on whether or not m_1 is a λ-abstraction or a Δ-abstraction and n'_1 is not, or m_1 and n'_1 are both a λ-abstraction or a Δ-abstraction. We only consider the non-trivial cases when $m_1 \equiv \lambda y : B'.m'_1$ and n'_1 is not a λ-abstraction, and $m_1 \equiv \Delta y : \neg(B' \to B).m'_1$ and n'_1 is not a Δ-abstraction. Consider the former.

Now by Lemma 8 it is the case that there exists a B'' such that $\mathsf{ctype}_A(x,t_1') = B''$, $B'' \equiv B' \to B$, and B is a subexpression of A, hence $A > B'$. By the definition of the hereditary substitution function $[n/x]^A(n_1' n_2') = [m_2/y]^{B'} m_1'$. Therefore, by the induction hypothesis there exists a normal form m such that $[m_2/y]^A m_1' = m$.

At this point consider when $m_1 \equiv \Delta y : \neg(B' \to B).m_1'$ and n_1' is not a Δ-abstraction. Again, by Lemma 8 it is the case that there exists a B'' such that $\mathsf{ctype}_A(x,t_1') = B''$, $B'' \equiv B' \to B$ and $B' \to B$ is a subexpression of A. Hence, $A > B'$. Let r be a fresh variable of type $\neg B$. Then by the induction hypothesis, there exists a term m'', such that, $\langle (y,r,m_2)\rangle_B^{B'} m_1' = m''$ and Therefore, $[n/x]^A(n_1' n_2') = \Delta r : \neg B.\langle (y,r,m_2)\rangle_B^{B'} m_1' = \Delta r : \neg B.m''$.

A.6 Proof of Soundness with Respect to Reduction

This is a mutually inductive proof using the lexicographic combination (A, f, t') of our ordering on types, the natural number ordering where $f \in \{0,1\}$, and the strict subexpression ordering on terms. We first prove part one and then part two. In both parts we case split on t'.

Part One.

Case. Suppose t' is a variable x. Then either there exists a term a such that $(x,z,a) \in \Theta$ or not. Suppose so. Then by definition we know $\langle\Theta\rangle^{\uparrow^A_{A'}} x = \lambda y : A \to A'.(z\,(y\,a))$, for some fresh variable y. Now $\langle\Theta\rangle^A_{A'} x = \lambda y : A \to A'.(z\,(y\,a))$, where we choose the same y. Thus, $\langle\Theta\rangle^{\uparrow^A_{A'}} x \rightsquigarrow^* \langle\Theta\rangle^A_{A'} x$. Now suppose there does not exist any term a or z such that $(x,z,a) \in \Theta$. Then $\langle\Theta\rangle^A_{A'} x = \langle\Theta\rangle^{\uparrow^A_{A'}} x = x$. Thus, $\langle\Theta\rangle^{\uparrow^A_{A'}} x \rightsquigarrow^* \langle\Theta\rangle^A_{A'} x$.

Case. Suppose $t' \equiv \lambda y : B_1.t_1'$. This case follows from the induction hypothesis.

Case. Suppose $t' \equiv \Delta y : \neg B.t_1'$. Similar to the previous case.

Case. Suppose $t' \equiv t_1' t_2'$. We have two cases to consider.

Case. Suppose $t_1' \equiv x$ for some variable x.

Case. Suppose $t_2' \equiv \lambda y : A.t_2''$, for some y and t_2'', $(x,z,t) \in \Theta$. Now

$$\begin{array}{rcl} \langle\Theta\rangle^{\uparrow^A_{A'}} (x\,(\lambda y : A.t_2'')) & = & (\lambda y : A \to A'.(z\,(y\,t)))\,(\lambda y : A.(\langle\Theta\rangle^{\uparrow^A_{A'}} t_2'')) \\ & \rightsquigarrow & z\,((\lambda y : A.(\langle\Theta\rangle^{\uparrow^A_{A'}} t_2''))\,t) \\ & \rightsquigarrow & z\,([t/y](\langle\Theta\rangle^{\uparrow^A_{A'}} t_2'')) \end{array}$$

Since $(A,1,t') > (A,1,t_2'')$ we can apply the induction hypothesis to obtain $\langle\Theta\rangle^{\uparrow^A_{A'}} t_2'' \rightsquigarrow^* \langle\Theta\rangle^A_{A'} t_2''$. Hence,

$$z\,([t/y](\langle\Theta\rangle^{\uparrow^A_{A'}} t_2'')) \quad \rightsquigarrow^* \quad z\,([t/y](\langle\Theta\rangle^A_{A'} t_2''))$$

Furthermore, sense $(A,1,t') > (A,0,\langle\Theta\rangle^A_{A'} t_2'')$, we also know from the induction hypothesis that

$$\begin{array}{rcl} z\,([t/y](\langle\Theta\rangle^A_{A'} t_2'')) & \rightsquigarrow^* & z\,([t/y]^A(\langle\Theta\rangle^A_{A'} t_2'')) \\ & = & \langle\Theta\rangle^A_{A'} t' \end{array}$$

Case. Suppose $t_2' \equiv \Delta y' : \neg(A \to A').t_2''$, for some y and t_2'', $(x,z,t) \in \Theta$. Now using a fresh variable z_2 we know

$$\begin{aligned}
&\langle\Theta\rangle^{\uparrow^A_{A'}}(x\,(\Delta y' : \neg(A \to A').t''_2)) \\
&\quad = (\lambda y : A \to A'.(z\,(y\,t)))\,(\Delta y' : \neg(A \to A').(\langle\Theta\rangle^{\uparrow^A_{A'}} t''_2)) \\
&\quad \rightsquigarrow z\,((\Delta y' : \neg(A \to A').(\langle\Theta\rangle^{\uparrow^A_{A'}} t''_2))\,t) \\
&\quad \rightsquigarrow z\,(\Delta z_2 : \neg A'.([\lambda y : A \to A'.(z_2\,(y\,t))/y'](\langle\Theta\rangle^{\uparrow^A_{A'}} t''_2))) \\
&\quad = z\,(\Delta z_2 : \neg A'.(\langle\Theta, (y', z_2, t)\rangle^{\uparrow^A_{A'}} t''_2))
\end{aligned}$$

Since $(A,1,t') > (A,1,t''_2)$ we know from the induction hypothesis that $\langle\Theta,(y',z_2,t)\rangle^{\uparrow^A_{A'}} t''_2 \rightsquigarrow^* \langle\Theta,(y',z_2,t)\rangle^A_{A'} t''_2$. Thus,

$$\begin{array}{rcl}
z\,(\Delta z_2 : \neg A'.(\langle\Theta,(y',z_2,t)\rangle^{\uparrow^A_{A'}} t''_2)) & \rightsquigarrow^* & z\,(\Delta z_2 : \neg A'.(\langle\Theta,(y',z_2,t)\rangle^A_{A'} t''_2)) \\
& = & \langle\Theta\rangle^A_{A'} t'.
\end{array}$$

Case. Suppose t'_2 is not an abstraction, and $(x,z,t) \in \Theta$. Since $(A,1,t') > (A,1,t'_2)$ we know from the induction hypothesis that $\langle\Theta\rangle^{\uparrow^A_{A'}} t'_2 \rightsquigarrow^* \langle\Theta\rangle^A_{A'} t'_2$. Thus,

$$\begin{array}{rcl}
\langle\Theta\rangle^{\uparrow^A_{A'}} t' & = & \langle\Theta\rangle^{\uparrow^A_{A'}}(x\,t'_2) \\
& = & z\,(\langle\Theta\rangle^{\uparrow^A_{A'}} t'_2) \\
& \rightsquigarrow^* & z\,(\langle\Theta\rangle^A_{A'} t'_2) \\
& = & \langle\Theta\rangle^A_{A'}(x\,t'_2) = \langle\Theta\rangle^A_{A'} t'.
\end{array}$$

Case. Suppose $(x,z,t'') \notin \Theta$ for any term t'' and z. Since $(A,1,t') > (A,1,t'_2)$ we know from the induction hypothesis that $\langle\Theta\rangle^{\uparrow^A_{A'}} t'_2 \rightsquigarrow^* \langle\Theta\rangle^A_{A'} t'_2$. Thus,

$$\begin{array}{rcl}
\langle\Theta\rangle^{\uparrow^A_{A'}} t' & = & \langle\Theta\rangle^{\uparrow^A_{A'}}(x\,t'_2) \\
& = & x\,(\langle\Theta\rangle^{\uparrow^A_{A'}} t'_2) \\
& \rightsquigarrow^* & x\,(\langle\Theta\rangle^A_{A'} t'_2) \\
& = & \langle\Theta\rangle^A_{A'}(x\,t'_2) = \langle\Theta\rangle^A_{A'} t'.
\end{array}$$

Case. Suppose t'_1 is not a variable. This case follows easily from the induction hypothesis.

Part two

Case. Suppose t' is a variable x or y distinct from x. Trivial in both cases.

Case. Suppose $t' \equiv \lambda y : B_1.s$. Then $[t/x](\lambda y : B_1.s) = \lambda y : B_1.([t/x]s)$. Now s is a strict subexpression of t' so we can apply the second part of the induction hypothesis to obtain $[t/x]s \rightsquigarrow^* [t/x]^A s$. At this point we can see that since $\lambda y : B_1.[t/x]s \equiv [t/x](\lambda y : B_1.s)$ we may conclude that $\lambda y : B_1.[t/x]s \rightsquigarrow^* \lambda y : B_1.[t/x]^A s$.

Case. Suppose $t' \equiv \Delta y : \neg B.s$. Similar to the previous case.

Case. Suppose $t' \equiv t'_1\,t'_2$. By Lemma 16 there exists terms s_1 and s_2 such that $[t/x]^A t'_1 = s_1$ and $[t/x]^A t'_2 = s_2$. Since t'_1 and t'_2 are strict subexpressions of t' we can apply the second part of the induction hypothesis to obtain $[t/x]t'_1 \rightsquigarrow^* s_1$ and $[t/x]t'_2 \rightsquigarrow^* s_2$. Now we case split on whether or not s_1 is a λ-abstraction and t'_1 is not, a Δ-abstraction and t'_1 is not, or s_1 is not a λ-abstraction or a Δ-abstraction. If s_1 is not a λ-abstraction or a Δ-abstraction then $[t/x]^A t' = ([t/x]^A t'_1)\,([t/x]^A t'_2) \equiv s_1\,s_2$. Thus, by two applications of the induction hypothesis, $[t/x]t' \rightsquigarrow^* [t/x]^A t'$, because $[t/x]t' = ([t/x]t'_1)\,([t/x]t'_2)$.

Suppose $s_1 \equiv \lambda y : B'.s'_1$ and t'_1 is not a λ-abstraction. By Lemma 8 there exists a type B'' such that $\mathsf{ctype}_A(x,t'_1) = B''$, $B'' \equiv B' \to B$, and B'' is a subexpression of A. Then by the definition of the hereditary substitution function $[t/x]^A(t'_1\,t'_2) = [s_2/y]^{B'} s'_1$. Now we know $A > B'$ so we can apply the second part of the induction hypothesis to obtain $[s_2/y]s'_1 \rightsquigarrow^* [s_2/y]^{B'} s'_1$. By knowing

that $((\lambda y : B'.s'_1)\, s_2) \rightsquigarrow ([s_2/y]s'_1)$ and by the previous fact we know $(\lambda y : B'.s'_1)\, s_2 \rightsquigarrow^* [s_2/y]^{B'} s'_1$. We now make use of the well known result of full β-reduction. The result is stated as

$$\frac{\begin{array}{ll} a \rightsquigarrow^* a' & \\ b \rightsquigarrow^* b' & a'\, b' \rightsquigarrow^* c \end{array}}{a\, b \rightsquigarrow^* c}$$

where a, a', b, b', and c are all terms. We apply this result by instantiating a, a', b, b', and c with $[t/x]t'_1$, s_1, $[t/x]t'_2$, s_2, and $[s_2/y]^{B'} s'_1$ respectively. Therefore, $[t/x](t'_1\, t'_2) \rightsquigarrow^* [s_2/y]^{B'} s'_1$.

Suppose $s_1 \equiv \Delta y : \neg(B' \to B).s'_1$ and t'_1 is not a Δ-abstraction. By Lemma 8 there exists a type B'' such that $\mathsf{ctype}_A(x, t'_1) = B''$, $B'' \equiv B' \to B$, and B'' is a subexpression of A. Then by the definition of the hereditary substitution function $[t/x]^A(t'_1\, t'_2) = \Delta z : \neg B.\langle (y, z, s_2)\rangle_B^{B'} s'_1$, where z is fresh variable. Now

$$\begin{array}{lcl} [t/x](t'_1\, t'_2) & = & ([t/x]t'_1)\,([t/x]t'_2) \\ & \rightsquigarrow^* & s_1\, s_2 \\ & \equiv & (\Delta y : \neg(B' \to B).s'_1)\, s_2 \\ & \rightsquigarrow & \Delta z : \neg B.[\lambda y' : B' \to B.(z\,(y'\, s_2))/y]s'_1 \\ & = & \Delta z : \neg B.(\langle (y, z, s_2)\rangle^{\uparrow_B^{B'}} s'_1) \end{array}$$

It suffices to show that $\Delta z : \neg B.(\langle (y, z, s_2)\rangle^{\uparrow_B^{B'}} s'_1) \rightsquigarrow^* \Delta z : \neg B.\langle (y, z, s_2)\rangle_B^{B'} s'_1$, but this follows from the induction hypothesis, because $(A, 0, t') > (B', 1, s'_1)$.

A.7 Proof of Type Soundness

This is a proof by induction on the assumed typing derivation.

Case.

$$\frac{}{\Gamma, x : A \vdash x : A}\ \textsc{Ax}$$

Trivial.

Case.

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \to B}\ \textsc{Lam}$$

By the induction hypothesis $t \in [\![B]\!]_{\Gamma, x:A}$. By the definition of the interpretation of types $t \rightsquigarrow^! n \in [\![B]\!]_{\Gamma, x:A}$ and $\Gamma, x : A \vdash n : B$. Thus, by applying the λ-abstraction type-checking rule, $\Gamma \vdash \lambda x : A.n : A \to B$, hence by the definition of the interpretation of types $\lambda x : A.n \in [\![A \to B]\!]_\Gamma$. Therefore, $\lambda x : A.t \rightsquigarrow^! \lambda x : A.n \in [\![A \to B]\!]_\Gamma$.

Case.

$$\frac{\Gamma, x : \neg A \vdash t : \bot}{\Gamma \vdash \Delta x : \neg A.t : A}\ \textsc{Delta}$$

Similar to the previous case.

Case.

$$\frac{\Gamma \vdash t_2 : A \qquad \Gamma \vdash t_1 : A \to B}{\Gamma \vdash t_1\, t_2 : B}\ \textsc{App}$$

By the induction hypothesis we know $t_1 \in [\![A \to B]\!]_\Gamma$ and $t_2 \in [\![A]\!]_\Gamma$. So by the definition of the interpretation of types we know there exists normal forms n_1 and n_2 such that $t_1 \rightsquigarrow^* n_1 \in [\![A \to B]\!]_\Gamma$ and $t_2 \rightsquigarrow^* n_2 \in [\![A]\!]_\Gamma$. Assume y is a fresh variable in n_1 and n_2 of type A. Then by hereditary substitution for the interpretation of types (Lemma 21) $[n_1/y]^A(y\,n_2) \in [\![B]\!]_\Gamma$. It suffices to show that $t_1\, t_2 \rightsquigarrow^* [n_1/y]^A(y\,n_2)$. This is an easy consequence of soundness with respect to reduction (Lemma 19), that is, $t_1\, t_2 \rightsquigarrow^* n_1\, n_2 = [n_1/y](y\,n_2)$ and by soundness with respect to reduction $[n_1/y](y\,n_2) \rightsquigarrow^* [n_1/y]^A(y\,n_2)$. Therefore, $t_1\, t_2 \in [\![B]\!]_\Gamma$.