# Mini Class on Normalization in Type Theory

*The Iowa Type Theory Commute*

Aaron Stump
Computer Science
The University of Iowa

November 3, 2021

## Contents

# 1 Introduction

The issue of whether or not programs terminate is of significant theoretical and practical importance in Computer Science. We will study one aspect of this topic, namely the normalization property for several different typed lambda calculi. Lambda calculus is a minimalistic programming language, whose central feature of anonymous functions is now found in most mainstream languages. We will review this below. Functional programming languages like LISP, Scheme, Haskell, and the various forms of ML (like OCaml) are based on lambda calculus, with Haskell and ML using statically typed versions.

Typed lambda calculus is also the core of proof assistants based on constructive type theory, like Coq and Agda. To ensure logical soundness of the theories implemented by these tools, it is necessary to ensure that all programs terminate. This is because, under the Curry-Howard isomorphism, proofs are identified with programs. For example, proofs by induction are identified with recursive programs: invoking the induction hypothesis in the proof is the same as making a recursive call. But then potentially diverging programs correspond to potentially unsound inductions, and must be banned by the system. Otherwise, one could prove any formula using a diverging function as the proof.

In this short class, we will see proofs of normalization properties for several different type theories, namely simply typed lambda calculus (STLC), System F (polymorphic lambda calculus), and System $F_\omega$ (a generalization of System F). These theories are described below. Let us begin with some basic definitions for typed lambda calculi.

# 2 Basics of Untyped Lambda Calculus

Lambda calculus expressions are called terms. The syntax for terms $t$ is:

$$\textit{terms}\ t \quad ::= \quad x \mid t\ t' \mid \lambda\, x.\, t$$

Here, $x$ is for variables, $t\ t'$ is for **applications** of $t$ as a function to $t'$ as an argument, and $\lambda\, x.\, t$ is a **lambda-abstraction**, an anonymous function which takes input $x$ and returns output $t$. The $\lambda$ in $\lambda\, x.t$ is said to bind $x$ in $t$. It introduces local variable $x$ within the **body** $t$ of the lambda abstraction.

(Note that we are here following the common practice of introducing a specific family of meta-variables – namely $t$ and variants like $t'$, $t_1$, etc. – to refer to a specific set of mathematical objects, in this case terms. Anywhere you see a $t$, you should understand that it is referring to some term as defined by the above syntax.)

## 2.1 $\beta$-reduction

To define how programs (terms) of lambda calculus run, the central idea is so-called $\beta$-reduction. It says that an anonymous function applied to an argument can be reduced by substituting the argument for the bound variable in the body of the function. This basic idea is expressed by defining a binary relation $\rightsquigarrow$ on terms by this rule:

$$\frac{}{(\lambda\, x.\, t)\ t' \ \rightsquigarrow\ [t'/x]t}\ \beta$$

$[t'/x]t$ denotes the term one gets by capture-avoiding substitution of $t'$ for $x$ in $t$. Capture-avoiding means that the scoping of variables should not change when doing the substitution. So substituting $x\ x$ for $y$ in $\lambda\, x.\, y$, for example, should not give us $\lambda x.\, (x\ x)$, because the scoping of $x$ in $x\ x$ is global, but would become local if the result were $\lambda x.\, (x\ x)$. Instead, one may rename the bound variable to avoid such capture, giving a result like $\lambda y.\, (x\ x)$ instead. It is actually rather tricky to get the details for substitution exactly right, both in theory and in practice. We will content ourselves with the idea that we must rename bound variables to avoid capture (even though this means that strictly speaking, substitution is now not a function, since more than one renaming is possible).

$$\frac{}{(\lambda\, x.\, t)\ t' \ \rightsquigarrow\ [t'/x]t}\ \beta \qquad \frac{t \ \rightsquigarrow\ t'}{\lambda\, x.\, t \ \rightsquigarrow\ \lambda\, x.\, t'}\ lam$$

$$\frac{t_1 \ \rightsquigarrow\ t_1'}{(t_1\ t_2) \ \rightsquigarrow\ (t_1'\ t_2)}\ app1 \qquad \frac{t_2 \ \rightsquigarrow\ t_2'}{(t_1\ t_2) \ \rightsquigarrow\ (t_1\ t_2')}\ app2$$

Figure 1: Rules defining full $\beta$-reduction

Terms which are the same if one safely (i.e., without capture) renames bound variables are called $\alpha$-equivalent.

Full $\beta$-reduction is the relation that allows the above $\beta$-reduction step to take place anywhere in a term. Since there can be more than *$\beta$-redex* (a term of the form $(\lambda\, x.\, t)\ t'$) in a term, full $\beta$-reduction is non-deterministic: it can happen that $t \rightsquigarrow t_1$ and $t \rightsquigarrow t_2$, where $t_1$ and $t_2$ are different. If for some similar relation we instead have that $t_1$ always equals $t_2$ in this situation, then the relation is *deterministic*. We will write $\rightsquigarrow^*$ for the reflexive-transitive closure of $\rightsquigarrow$. We have $t \rightsquigarrow^* t'$ iff there is a finite sequence of $\rightsquigarrow$-steps leading from $t$ to $t'$. In other words, we allow 0 or more steps of $\beta$-reduction.

The relation of full $\beta$-reduction may be defined using the rules of Figure 1. Here is an example derivation using the rules above to show that $\lambda x.x\ ((\lambda z.z\ z)\ x)$ reduces to $\lambda x.x\ (x\ x)$.

$$\dfrac{\dfrac{\dfrac{}{(\lambda\, z.\, (z\ z))\ x \ \rightsquigarrow x\ x}\ \beta}{x\ ((\lambda\, z.\, (z\ z))\ x) \ \rightsquigarrow\ x\ (x\ x)}\ app2}{\lambda\, x.\, (x\ ((\lambda z.z\ z)\ x)) \ \rightsquigarrow\ \lambda\, x.\, (x\ (x\ x))}\ lam$$

Generally, derivations are trees labeled by judgments of some kind (facts to be proved by the rules). The leaves are drawn at the top, and the root at the bottom. Edges between tree nodes are indicated with horizontal lines, showing that an inference has been made using a particular rule. Axioms are rules that have no premises, and hence may be placed used at the leaves.

## 2.2 Example of $\beta$-reduction

Let $id$ abbreviate the term $\lambda\, x.\, x$, and if $n$ is a natural number ($\{0,1,\dots\}$), let $\dot{n}$ abbreviate

$$\lambda\, s.\, \lambda\, z.\, \underbrace{(s \cdots (s}_{n}\ z) \cdots)$$

This is the Church-encoding of number $n$. Then for example, we have

$$\dot{n}\ id \rightsquigarrow^*\ id$$

This is because the whole term $\dot{n}\ id$ itself is a $\beta$-redex (lambda abstraction applied to argument), which reduces to

$$\lambda\, z.\, \underbrace{(id \cdots (id}_{n}\ z) \cdots)$$

Reducing (sequentially) the $n$ applications of $id$ results in $\lambda\, z.\, z$, which is indeed $\alpha$-equivalent to $id$. This is an artificial example, but one can define the usual arithmetic operations like addition, multiplication, etc., using the Church encoding.

**Exercises:**

1. Write down a reduction sequence (i.e., a sequence of $\beta$-reductions like $t_1 \rightsquigarrow t_2 \rightsquigarrow \cdots$) from the following term to a normal form:
$$(\lambda\, f.\, \lambda\, a.\, f\ (f\ a))\ (\lambda\, x.\, \lambda\, y.\, x)$$
2. For the previous example, how many different reduction sequences are there to a normal form?

## 2.3 Call-by-name reduction

Functional programming languages generally do not implement full $\beta$-reduction for their lambda calculus fragments, but rather some **reduction strategy**. This is a particular choice, for each term, of a subset of its $\beta$-redexes that may be reduced next. Below, we will show some of our normalization results for a strategy called **call-by-name**. This is a form of lazy reduction, where we do not require arguments $t'$ to be reduced before reducing $(\lambda\, x.\, t)\ t'$. We can define single-step call-by-name reduction with these inference rules:

$$\frac{}{(\lambda\, x.\, t)\ t' \ \rightsquigarrow\ [t'/x]t}\ \beta \qquad \frac{t_1 \ \rightsquigarrow\ t_1'}{t_1\ t_2 \ \rightsquigarrow\ t_1'\ t_2}\ \mathit{left}$$

Rather than just stipulating, as we did for full $\beta$-reduction, that any redex we find can be reduced, we here require a finite derivation of $t \rightsquigarrow t'$ using these rules. The first rule is just the $\beta$ rule again, of course. The second, here called *left*, quite restricts where that $\beta$ rule can be applied. It can only be applied in the left part of an application (or, using the $\beta$ rule directly, at the top level of the term). A derivation built using these two rules has a use of the $\beta$-rule at the sole leaf of tree, followed by some sequence of uses of the *left*-rule. Here is an example derivation:

$$\frac{\dfrac{}{(\lambda\, x.\, \lambda\, y.\, \lambda z.\, (y\ x))\ (id\ id) \ \rightsquigarrow\ (\lambda\, y.\, \lambda\, z.\, (y\ (id\ id)))}\ \beta}{(\lambda\, x.\, \lambda\, y.\, \lambda z.\, (y\ x))\ (id\ id)\ id \ \rightsquigarrow\ (\lambda\, y.\, \lambda\, z.\, (y\ (id\ id)))\ id}\ \mathit{left}$$

This derivation proves the single reduction step (at the bottom of the derivation)

$$(\lambda\, x.\, \lambda\, y.\, \lambda z.\, (y\ x))\ (id\ id)\ id \ \rightsquigarrow\ (\lambda\, y.\, \lambda\, z.\, (y\ (id\ id)))\ id$$

We can write a call-by-name *reduction sequence* beginning with this step:

$$(\lambda\, x.\, \lambda\, y.\, \lambda z.\, (y\ x))\ (id\ id)\ id \ \rightsquigarrow\ (\lambda\, y.\, \lambda\, z.\, (y\ (id\ id)))\ id \ \rightsquigarrow\ \lambda\, z.\, (id\ (id\ id))$$

The second reduction also has a derivation, just using the $\beta$-rule. The final term of the reduction sequence – i.e., $\lambda\, z.\, (id\ (id\ id))$ – is a *normal form* with respect to call-by-name reduction: it cannot be reduced further with this relation. This is because the rules do not allow reduction underneath a $\lambda$-abstraction or in the right part of an application, since the only rule for finding a $\beta$-redex is *left*. This term is not a normal form with respect to full $\beta$-reduction, however, because it has two $\beta$-redexes in it, namely the two applications of *id*. This shows that normalization with call-by-name reduction might result in a term that has $\beta$-redexes in it, in the body of a $\lambda$-abstraction. It could also happen, if one has free (undeclared) variables in the term, that there could be a $\beta$-redex in the right part of an application of a variable, as in $x\ (id\ id)$. But if the term is **closed** (no occurrences of variable $x$ except under some $\lambda$ that binds $x$), then terms which are normal with respect to call-by-name reduction are $\lambda$-abstractions. These might contain $\beta$-redexes in their bodies.

Call-by-name reduction is a deterministic strategy.

## 2.4 The Simply Typed Lambda Calculus

The Simply Typed Lambda Calculus (STLC) is a basic type system that one finds as a subsystem of most practical type systems. It is thus of interest. Type systems always come with some set of types. The simple types $T$ are defined over some unspecified set of base types (in practice, these could be types like `char` or `int`):

$$\begin{array}{rcl} \textit{base types } b & & \\ \textit{simple types } T & ::= & b \mid T_1 \to T_2 \end{array}$$

In these notes, we use a Curry-style approach to typing, where the type system specifies assignments of types to unannotated terms of pure $\lambda$-calculus. For example, intuitively, the term *id* can be assigned any type

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T} \quad \frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \lambda\, x.\, t : T_1 \to T_2} \quad \frac{\Gamma \vdash t_1 : T_2 \to T_1 \quad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1\ t_2 : T_1}$$

Figure 2: Type-assignment rules for simply typed lambda calculus

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{x : T_1 \to T_2, y : T_1 \vdash x : T_1 \to T_2} \quad \overline{x : T_1 \to T_2, y : T_1 \vdash y : T_1}}{x : T_1 \to T_2, y : T_1 \vdash (x\ y) : T_2}}{x : T_1 \to T_2 \vdash \lambda\, y.\, (x\ y) : T_1 \to T_2}}{\cdot \vdash \lambda\, x.\, \lambda\, y.\, (x\ y) : (T_1 \to T_2) \to (T_1 \to T_2)}$$

Figure 3: Example typing derivation in STLC

of the form $T \to T$; for example, $b \to b$. Church-style typing is more common, where terms are annotated with types – most commonly, for $\lambda$-bound variables – and these annotations are treated as essential parts of the term. It is perfectly reasonable to use annotations with Curry-style typing, as hints to guide the type checker in applying the typing rules, which might be nondeterministically applicable. For theoretical study, though, it is generally more convenient to disregard the question of interpreting typing rules as a description of a typing algorithm, and instead just view them as defining a mathematical relation of typing between terms and types.

Actually, there is usually a third component for a typing judgment (that is, a statement of typing that is to be derived using the typing rules). This is the *typing context*, which is a list of assumptions of types for variables:

$$\textit{typing contexts}\ \Gamma \quad ::= \quad \cdot \mid \Gamma, x : T$$

The empty context, which declares types for no variables, is written $\cdot$. If $\Gamma$ is a context, then adding an assumption that term variable $x$ has type $T$ is denoted $\Gamma, x : T$. The type-assignment rules for STLC are shown in Figure 2. In the first rule, for typing variables, the notation $\Gamma(x) = T$ is used to mean that the result of looking up the first type for variable $x$ in context $\Gamma$, starting from the right, is $T$.

An example typing derivation is given in Figure 3. It shows that in the empty context, $\lambda\, x.\, \lambda\, y.\, (x\ y)$ can be assigned the type $(T_1 \to T_2) \to (T_1 \to T_2)$, for any types $T_1$ and $T_2$.

**Exercises:** Write out typing derivations, using the rules of Figure 2, for the following typing judgments, assuming base types $a$, $b$, and $c$.

1. $\cdot, x : b, y : b \to b \vdash y\ (y\ x) : b$
2. $\cdot \vdash \lambda\, x.\, \lambda\, y.\, x : a \to (b \to a)$
3. $\cdot \vdash \lambda\, x.\, \lambda\, y.\lambda\, z.\, ((x\ z)\ (y\ z)) : (a \to (b \to c)) \to ((a \to b) \to (a \to c))$

# 3 Normalization for the Simply Typed Lambda Calculus

In this section, we will prove that terms that are statically typable in a particular type system called the Simply Typed Lambda Calculus (STLC) are normalizing with respect to full $\beta$-reduction. Since $\beta$-reduction is nondeterministic, an important issue is whether we will prove that no matter which redex one chooses to reduce next, reduction reaches a normal form; or instead that there exists a particular choice of redex that guarantees reduction reaches a normal form (but where it is conceivable that reduction could diverge if a different strategy were applied). Reduction no matter what strategy is used is called **strong**

**normalization** (also **termination**), while reduction under some strategy (but possibly not all) is called **weak normalization**, or just normalization. Here we will prove just weak normalization.

Quite a simple – and to me, appealing – proof of normalization for terms typable in STLC can be given using a well-founded ordering invented by Dershowitz and Manna [1979]. The idea of the normalization proof is to define, for each typing derivation, a certain measure. This measure will be a multiset of types, one for each redex we find in the typing derivation. Then we will see a particular reduction strategy (choice of next redex to reduce) that always causes this measure to decrease in the Dershowitz-Manna ordering. The ordering is well-founded, which means that one cannot decrease forever. So eventually we will reach a minimal multiset of types. In this case, the minimal multiset will be the empty one, as all redexes will have been reduced.

There are several ingredients to this proof, which we consider in turn.

## 3.1 A measure of a typing derivation

Suppose $\mathcal{D}$ is a typing derivation for judgment $\Gamma \vdash t : T$. Then define a measure $\mu(\mathcal{D})$ to be the multiset of types $T_2 \to T_1$ which occur in an inference of the following form in $\mathcal{D}$:

$$\dfrac{\dfrac{\Gamma, x : T_2 \vdash t : T_1}{\Gamma \vdash (\lambda x.t) : T_2 \to T_1} \quad \Gamma \vdash t' : T_2}{\Gamma \vdash (\lambda x.t)\ t' : T_1}$$

This inference is typing a $\beta$-redex $(\lambda x.t)\ t'$ by typing the $\lambda$-abstraction at type $T_2 \to T_1$. For all such typings of $\beta$-redexes in $\mathcal{D}$, the multiset $\mu(\mathcal{D})$ contains the type $T_2 \to T_1$. Let us call such a type a *redex type* of $\mathcal{D}$.

For example, consider this typing derivation $\mathcal{D}$ for the term $\lambda\, x.\, (id\ (id)\ x)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot, x : b, x : b \vdash x : b}}{\cdot, x : b \vdash id : \mathbf{b \to b}} \quad \dfrac{\dfrac{\dfrac{}{\cdot, x : b, x : b \vdash x : b}}{\cdot, x : b \vdash id : \mathbf{b \to b}} \quad \dfrac{}{\cdot, x : b \vdash x : b}}{\cdot, x : b \vdash id\ x : b}}{\cdot, x : b \vdash (id\ (id\ x)) : b}}{\cdot \vdash \lambda\, x.\, (id\ (id\ x)) : b \to b}$$

The multiset $\mu(\mathcal{D})$ is $\{b \to b, b \to b\}$, because those types are included in the multiset at the bolded points in the derivation. (Recall that a multiset is like a set except that multiplicity is taken into account.) Let us say that we are just ordering types by their size.

## 3.2 The Dershowitz-Manna multiset ordering

Dershowitz and Manna defined a very useful and influential ordering on finite multisets of elements of type $A$, extending a strict partial order (transitive and irreflexive) on $A$ [Dershowitz and Manna, 1979]. The definition is that a finite multiset $M$ is greater than $M'$ (notation $M \gg M'$) iff there are finite multisets $X$ and $Y$ such that

$$M' = (M - X) \cup Y$$

and further, for all $y \in Y$, there is some $x \in X$ such that $x > y$. Dershowitz and Manna prove that this is a strict partial order, and that it is well-founded iff the ordering on $A$ is.

For example, suppose that $A$ is the set of natural numbers, and we use the usual decreasing ordering on those. Then a multiset like $\{5, 5, 3, 1, 1\}$ is greater than $\{4, 4, 4, 4, 5, 3, 1, 0, 0, 0\}$. Here, the multiset $X$ is $\{5, 1\}$ and the multiset $Y$ is $\{4, 4, 4, 4, 0, 0, 0\}$. For every element of $Y$, there is indeed a greater element of $X$. The power of the ordering comes from the fact that there may be (finitely) many elements of $Y$ dominated by a single element of $X$.

### 3.3 Type preservation: reduction preserves typing

Our measure is going to be defined on typing derivations, but reduction operates on terms of pure $\lambda$-calculus. We must prove that we can keep typing and reduction in synch: whenever we have $t \rightsquigarrow t'$ and $\Gamma \vdash t : T$, then we also have $\Gamma \vdash t' : T$. This property is called *type preservation*.

**Theorem 1** (Type Preservation). *If* $\Gamma \vdash t : T$ *and* $t \rightsquigarrow t'$, *then* $\Gamma \vdash t' : T$.

This is a straightforward and intuitive property, which one generally expects to hold for most type systems. The proof is given in Section A in the Appendix. The proof makes use of the following lemma, also proved in the Appendix:

**Lemma 2** (Substitution). *If* $\Gamma, x : T_2 \vdash t : T_1$ *and* $\Gamma \vdash t' : T_2$, *then* $\Gamma \vdash [t'/x]t : T_1$.

This lemma says that substituting a term of type $T_2$ for a variable that is assumed to have type $T_2$ preserves typing (while eliminating that variable from the typing context). This plays a crucial role in the next section.

### 3.4 A reduction strategy that reduces the measure

The transformation implicit in the (constructive) proof of Theorem 1 could change the measure of a typing derivation only in the following situation. The proof of $t \rightsquigarrow t'$ ends in

$$\frac{}{(\lambda\, x.\, t)\ t' \ \rightsquigarrow\ [t'/x]t}\ \beta$$

and the typing derivation ends in:

$$\frac{\dfrac{\Gamma, x : T_2 \vdash t : T_1}{\Gamma \vdash (\lambda x.t) : T_2 \to T_1} \quad \Gamma \vdash t' : T_2}{\Gamma \vdash (\lambda x.t)\ t' : T_1} \tag{A}$$

In this case, the proof of Theorem 1 says to transform this derivation into

$$\frac{\Gamma, x : T_2 \vdash t : T_1 \quad \Gamma \vdash t' : T_2}{\Gamma, \vdash [t'/x]t : T_1}\ \textit{Lemma 2} \tag{B}$$

(That is, we use the derivation which the proof of Lemma 2 constructs for the typing judgment with $[t'/x]t$). We need to ensure that the measure of this derivation is smaller, in the Dershowitz-Manna ordering, than the measure of the typing derivation we started with. We are going to remove one copy of $T_2 \to T_1$ from the measure, when we transform (A) into (B). For the measure to decrease, we need to make sure that the only types we add to the multiset are smaller than $T_2 \to T_1$. This would be a problem in the situation where the typing derivation for $t'$ contained a redex type of the same size or larger than $T_2 \to T_1$, and the variable $x$ occurred multiple times in $t$. For then the proof of Lemma 2 would copy the typing derivation for $t'$ several times, and that redex type would get added additional times to the multiset. The elimination of one copy of $T_2 \to T_1$ would not suffice to cover those copied redex types (since we are assuming the copied redex types are not smaller than $T_2 \to T_1$).

We can now see how to define our reduction strategy for terms with typing derivation $\mathcal{D}$. We may reduce redexes $(\lambda\, x.\, t)\ t'$ as long as either $x$ does not occur more than once in $t$, or else the redex type in $\mathcal{D}$ of the redex is greater than the redex types of the redexes (if any) in the argument $t'$. Redexes of smaller redex type are allowed to be copied – but these copies are then covered by the elimination of the redex type of $(\lambda\, x.\, t)\ t'$.

There is one final step of reasoning then required to show:

**Theorem 3.** *Suppose $\mathcal{D}$ derives $\Gamma \vdash t : T$, and $t \rightsquigarrow t'$ using the strategy just described. Let $\mathcal{D}'$ be the derivation stated to exist by Theorem 1. Then $\mu(\mathcal{D}) \gg \mu(\mathcal{D}')$.*

*Proof.* The final important consideration is that when we perform a $\beta$-reduction, we might create new redexes that were not present before in the term. For example, in the following reduction, we create the redex *id id*:

$$(\lambda\, x.\, (x\ id))\ id\ \rightsquigarrow id\ id$$

The redex is created in the sense that the term it comes from – namely, $(x\ id)$ – is not a redex, but it is a redex. So we have to worry about our ordering decreasing in this case. When we have a redex with redex type $T_2 \to T_1$, we are substituting an argument term of smaller type $T_2$ for the bound variable of the $\lambda$-abstraction. Following the proof of Lemma 2, we will find that the redex type of redexes created by the substitution can only be $T_2$. The addition of several redexes with redex type $T_2$ is covered, again thanks to the Dershowitz-Manna ordering, by the removal of the redex type $T_2 \to T_1$. □

**Corollary 4.** *If $\Gamma \vdash t : T$, then $t$ is normalizing.*

*Proof.* By the preceding theorem, if we reduce $t$ using the strategy that only reduces redexes $r$ whose redex type (in the associated typing derivation) is greater than the redex types of any redexes strictly within $r$, we will reduce the measure on the corresponding typing derivation. Since the ordering in question is well-founded (since the ordering on types is), we must eventually reach a term $t'$ which cannot be further reduced using the strategy. It suffices just to show that $t'$ really is a normal form. For sake of contradiction, let $r$ be the smallest redex in $t'$. Since it is smallest, it could be reduced by our strategy, since by definition it does not have any redexes within in it (and hence cannot have any with redex type of greater or equal size). This contradicts the assumption that $t'$ cannot be further reduced with our strategy. □

**Exercises:**

1. Underline all the redexes in this term:

$$\cdot \vdash (\lambda\, q.\, \lambda\, h.\, ((\lambda\, z.\, (q\ z\ z))\ ((\lambda\, y.\, y\, h)\ (\lambda\, x.\, x)))) \ (\lambda\, u.\, \lambda\, v.\, u)$$

2. Confirm (informally, or with a full typing derivation) that this term can be assigned type $b$ if we make these assignments for the bound vvariables:
   - $u : b$
   - $v : b$
   - $h : b$
   - $q : b \to (b \to b)$
   - $z : b$
   - $y : b \to b$
   - $x : b$
3. What are the redex types given this assignment of types to the bound variables?
4. Is there any redex that is not allowed by the above reduction strategy?
5. Show one reduction sequence that is allowed by our strategy, and which reaches a normal form. Show the measure for the typing derivation associated with each term in the sequence (to confirm it is indeed decreasing).

# 4 Call-by-name Normalization for System F

This section gives a proof that call-by-name reduction is normalizing for unannotated System F (polymorphic lambda calculus). System F is defined with annotated terms, where $\lambda$-bound variables must be declared with their types. So we have $\lambda x : T.t$ instead of just $\lambda x.t$. For metatheoretic analysis, I prefer to work with unannotated terms. This system (with unannotated terms) is also called $\lambda 2$.

In addition to proving normalization, we will be able to draw an easy corollary:

Since System F includes STLC, the proof given here can be viewed as a different way to prove a normalization property just for STLC (just ignore the parts of the proof dealing with polymorphism). But we fundamentally need a different approach to normalization for System F, because there is no known way to extend the measure we used in the proof of the previous section, to work with the *impredicative* polymorphism of System F. Impredicativity means that types may quantify over all types, including themselves. In contrast, predicative definitions may only reference sets of previously constructed elements.

With impredicative polymorphism, the linchpin in the proof above of Theorem 3 fails. If we have a redex whose redex type is of the form $(\forall X.T) \to T'$, then when we substitute the argument term for the bound variable of type $\forall X.T$, we may end up creating a redex with a much bigger redex type. This is because the variable might be applied to an argument in the body of the $\lambda$-abstraction, after instantiating its type variable $X$ with some large type.

The method of defining a compositional interpretation of simple types in order to show normalization is due to Tait [1967]. It was extended by Girard to handle impredicative polymorphism [Girard, 1972]. An accessible source for Girard's proof is Girard et al. [1989], which I recommend for learning more about these topics.

Proving just that call-by-name reduction normalizes is simpler than proving normalization of full $\beta$-reduction, in one important case. For a $\lambda$-abstraction of type $T_1 \to T_2$, it could happen that $T_1$ is an empty type (i.e., no terms have that type). Then it requires some extra work to make sure that the body of the $\lambda$-abstraction normalizes, even when a variable of empty type is in scope. It is simpler not to have to worry about that case. Since call-by-name reduction does not descend under $\lambda$-abstractions, we can avoid the issue, at the cost of a weaker theorem: when reducing a closed term, $\beta$-redexes can still occur in the final term, but only under $\lambda$-abstractions.

## 4.1 Syntax

| | | |
|---|---|---|
| *term variables* $x$ | | |
| *type variables* $X$ | | |
| *terms* $t$ | $::=$ | $x \mid \lambda x.t \mid t\ t'$ |
| *types* $T$ | $::=$ | $X \mid T \to T' \mid \forall\, X.\, T$ |

The new form of type is $\forall\, X.\, T$. This is a polymorphic type, which intuitively means that for any type $T'$, $t$ can be typed with the instance $[T'/X]T$.

## 4.2 Typing

A typing context $\Gamma$ declares free term and type variables:

$$\textit{Typing context}\ \Gamma ::= \cdot \mid \Gamma, x : T \mid \Gamma, X : \star$$

We write $\Gamma(x) = T$ to mean looking up the first typing for $x$ in $\Gamma$, starting from the right. The typing rules are in Figure 4. To ensure that types are well-formed, we use some extra rules, called *kinding* rules, in Figure 5.

$$\frac{\Gamma(x) = T \quad \Gamma \vdash T : \star}{\Gamma \vdash x : T} \qquad \frac{\Gamma, x : T \vdash t : T'}{\Gamma \vdash \lambda x.t : T \to T'} \qquad \frac{\Gamma \vdash t : T_1 \to T_2 \quad \Gamma \vdash t' : T_1}{\Gamma \vdash t\ t' : T_2}$$

$$\frac{\Gamma, X : \star \vdash t : T}{\Gamma \vdash t : \forall X.T} \qquad \frac{\Gamma \vdash t : \forall X.T \quad \Gamma \vdash T' : \star}{\Gamma \vdash t : [T'/X]T}$$

Figure 4: Typing rules for unannotated System F

$$\frac{\Gamma(X) = \star}{\Gamma \vdash X : \star} \qquad \frac{\Gamma \vdash T_1 : \star \quad \Gamma \vdash T_2 : \star}{\Gamma \vdash T_1 \to T_2 : \star} \qquad \frac{\Gamma, X : \star \vdash T : \star}{\Gamma \vdash \forall X.T : \star}$$

Figure 5: Kinding rules for unannotated System F

An example derivable typing is

$$\cdot \vdash \lambda\, x.\, (x\ x) : \mathit{Unit} \to \mathit{Unit}$$

where *Unit* abbreviates $\forall\, X.\, (X \to X)$. The idea is that if $x$ has type *Unit*, then we may apply it to an argument of type $X$, and get back a result of type $X$, for any $X$. So we may apply $x$ to itself, by instantiating $X$ with *Unit*. (So where we are using $x$ here as a function, we are using it at type $\mathit{Unit} \to \mathit{Unit}$.)

The proof of the following theorem relating typing and kinding is straightforward and omitted. The name "regularity" for the property expressed by the theorem is standard.

**Theorem 5** (Regularity)**.** *If* $\Gamma \vdash t : T$ *then* $\Gamma \vdash T : \star$.

## 4.3 Semantics for types

Figure 6 gives a compositional semantics $[\![T]\!]_\rho$ for types. (This notation for semantic value is widely used in Logic, Linguistics, and Computer Science, and is due to Dana Scott [Rabern, 2016].) The function $\rho$ gives the interpretations of free type variables in $T$. Each free type variable is interpreted as a *reducibility candidate*, and we write $\rho$ only for functions mapping type variables $X$ to reducibility candidates. To define what a reducibility candidate is: let us denote the set of closed terms which normalize using call-by-name reduction as $\mathcal{N}$. We will write $\rightsquigarrow$ for call-by-name reduction. Then a reducibility candidate $R$ is a set of terms satisfying the following requirements:

- $R \subseteq \mathcal{N}$
- If $t \in R$ and $t' \rightsquigarrow t$, then $t' \in R$

In other words, it is a set of normalizing terms closed under $\beta$-expansion (the converse relation of $\beta$-reduction). The set of all reducibility candidates is denoted $\mathcal{R}$.

**Lemma 6** ($\mathcal{R}$ is a cpo)**.** *The set* $\mathcal{R}$ *ordered by subset forms a complete partial order: it has greatest element* $\mathcal{N}$*, and greatest lower bound of a nonempty set of elements of* $\mathcal{R}$ *given by intersection.*

*Proof.* $\mathcal{N}$ satisfies both requirements for a reducibility candidate, and since one of those requirements is being a subset of $\mathcal{N}$, it is clearly the largest such set to do so. Let us prove that the intersection of a nonempty set $S$ of reducibility candidates is still a reducibility candidate. Certainly if the members of $S$ are subsets of $\mathcal{N}$ then so is $\bigcap S$. For the second property: assume an arbitrary $t \in \bigcap S$ with $t' \rightsquigarrow t$, and show $t' \in \bigcap S$. For the latter, it suffices to show $t' \in R$ for every $R \in S$. Consider an arbitrary such $R$. From $t \in \bigcap S$ and $R \in S$, we have $t \in R$. Then since $R$ is expansion-closed (since it is a reducibility candidate), $t \in R$ and $t' \rightsquigarrow t$ implies $t' \in R$, . □

$$
\begin{array}{lcl}
[\![X]\!]_\rho & = & \rho(X) \\
[\![T_1 \to T_2]\!]_\rho & = & \{t \in \mathcal{N} \mid \forall t' \in [\![T_1]\!]_\rho.\ t\ t' \in [\![T_2]\!]_\rho\} \\
[\![\forall\, X.\, T]\!]_\rho & = & \bigcap_{R \in \mathcal{R}} [\![T]\!]_{\rho[X \mapsto R]}
\end{array}
$$

Figure 6: Reducibility semantics for types

**Lemma 7** (The semantics of types computes reducibility candidates)**.** *If $\rho(X)$ is defined for every free type variable of $T$, then $[\![T]\!]_\rho \in \mathcal{R}$.*

*Proof.* The proof is by induction on the structure of the type. If $T$ is a type variable $X$, then by assumption, $\rho(X)$ is a reducibility candidate, and this is the value of $[\![T]\!]_\rho$.

If $T$ is an arrow type $T_1 \to T_2$, we must prove the two properties listed above for being a reducibility candidate. Certainly $[\![T]\!]_\rho \subseteq \mathcal{N}$, because the semantics of arrow types requires this explicitly. Now suppose that $t \in [\![T_1 \to T_2]\!]_\rho$ and $t' \rightsquigarrow t$. We must show $t' \in [\![T_1 \to T_2]\!]_\rho$. Since $t$ is normalizing and $t' \rightsquigarrow t$, we know that $t'$ is also normalizing (there is a reduction sequence from $t'$ to $t$ and from $t$ to a normal form). So let us assume an arbitrary $t'' \in [\![T_1]\!]_\rho$, and show that $t'\ t'' \in [\![T_2]\!]_\rho$. Since $t' \rightsquigarrow t$, by the definition of call-by-name reduction, we have

$$t'\ t'' \rightsquigarrow t\ t''$$

Since $t \in [\![T_1 \to T_2]\!]_\rho$, we know by the semantics of types that $t\ t'' \in [\![T_2]\!]_\rho$, since $t'' \in [\![T_1]\!]_\rho$. By the IH, $[\![T_2]\!]_\rho$ is a reducibility candidate. So since $t'\ t'' \rightsquigarrow t\ t''$ and $t\ t'' \in [\![T_2]\!]_\rho$, we also have $t'\ t'' \in [\![T_2]\!]_\rho$. This was all we had to prove in this case.

Finally, if $T$ is a universal type $\forall\, X.\, T'$, then by IH, the set $[\![T']\!]_{\rho[X \mapsto R]}$ is a reducibility candidate for all $R \in \mathcal{R}$. Since $\mathcal{R}$ is a complete partial order, $\bigcap_{R \in \mathcal{R}} [\![T']\!]_{\rho[X \mapsto R]}$ is then also a reducibility candidate.

□

## 4.4 Soundness of Typing Rules

The goal of this section is to prove that terms which can be assigned a type using the rules of Figure 4 are normalizing. We will actually prove a stronger statement, based on an interpretation of typing judgments. First, we must define an interpretation $[\![\Gamma]\!]$ for typing contexts $\Gamma$. This interpretation will be a set of pairs $(\sigma, \rho)$, where $\rho$ is, as above, a function mapping type variables to reducibility candidates; and $\sigma$ maps term variables to terms. The definition is by recursion on the structure of $\Gamma$:

$$
\begin{array}{lcl}
(\sigma, \rho) \in [\![x : T, \Gamma]\!] & \Leftrightarrow & \sigma(x) \in [\![T]\!]_\rho \ \wedge\ (\sigma, \rho) \in [\![\Gamma]\!] \\
(\sigma, \rho) \in [\![X : *, \Gamma]\!] & \Leftrightarrow & \rho(x) \in \mathcal{R} \ \wedge\ (\sigma, \rho) \in [\![\Gamma]\!] \\
(\sigma, \rho) \in [\![\cdot]\!] & &
\end{array}
$$

In the statement of the theorem below, we write $\sigma t$ to mean the result of simultaneously substituting $\sigma(x)$ for $x$ in $t$, for all $x$ in the domain of $\sigma$.

**Lemma 8.** *Suppose $(\sigma, \rho) \in [\![\Gamma]\!]$. If $t \in [\![T]\!]_\rho$, then $(\sigma[x \mapsto t], \rho) \in [\![\Gamma, x : T]\!]$. Also, if $R \in \mathcal{R}$, then $(\sigma, \rho[x \mapsto R]) \in [\![\Gamma, X : *]\!]$.*

*Proof.* The proof of the first part is by induction on $\Gamma$. If $\Gamma = \cdot$, then to show $(\sigma[x \mapsto t], \rho) \in [\![\cdot, x : T]\!]$, it suffices to show $t \in [\![T]\!]_\rho$, which holds by assumption. If $\Gamma = y : T', \Gamma'$, then we have $(\sigma, \rho) \in [\![\Gamma']\!]$ by the definition of $[\![\Gamma]\!]$, and we may apply the IH to conclude $(\sigma[x \mapsto t], \rho) \in [\![\Gamma', x : T]\!]$, from which we can conclude the desired $(\sigma[x \mapsto t], \rho) \in [\![\Gamma, x : T]\!]$, again by the definition of $[\![\Gamma]\!]$. Similar reasoning applies if $\Gamma = X : \star, \Gamma'$. The proof of the second part of the lemma is exactly analogous. □

**Theorem 9** (Soundness of typing rules with respect to the semantics)**.** *If* $\Gamma \vdash t : T$*, then for all* $(\sigma, \rho) \in [\![\Gamma]\!]$*, we have* $\sigma t \in [\![T]\!]_\rho$*.*

*Proof.* The proof is by induction on the structure of the assumed typing derivation. In each case, we will implicitly assume an arbitrary $(\sigma, \rho) \in [\![\Gamma]\!]$. Let us start with an easy case that shows the power of Tait's semantics for

Case:

$$\frac{\Gamma \vdash t : T_1 \to T_2 \quad \Gamma \vdash t' : T_1}{\Gamma \vdash t\ t' : T_2}$$

By the IH, $\sigma t \in [\![T_1 \to T_2]\!]_\rho$ and $\sigma t' \in [\![T_1]\!]_\rho$. By the semantics of arrow types, this immediately implies $(\sigma t)\ (\sigma t') \in [\![T_2]\!]_\rho$, as required.

Case:

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T}$$

We proceed by inner induction on $\Gamma$. If $\Gamma$ is empty, then $\Gamma(x) = T$ is false, and this case cannot arise. Suppose $\Gamma$ is of the form $x : T, \Gamma'$. Then $\sigma(x) \in [\![T]\!]_\rho$ by definition of $[\![\Gamma]\!]$, which suffices to prove the conclusion. Suppose $\Gamma$ is of the form $y : T, \Gamma'$, where $y \neq x$, or of the form $X : *, \Gamma'$. Then $\Gamma'(x) = T$ and $(\sigma, \rho) \in [\![\Gamma']\!]$, and we use the induction hypothesis to conclude $\sigma x \in [\![T]\!]_\rho$.

Case:

$$\frac{\Gamma, x : T \vdash t : T'}{\Gamma \vdash \lambda x.t : T \to T'}$$

All closed $\lambda$-abstractions are in $\mathcal{N}$. So to prove $(\lambda x.\sigma t) \in [\![T \to T']\!]_\rho$, it suffices to assume an abitrary $t' \in [\![T]\!]_\rho$ and prove $(\lambda x.\sigma t)\ t' \in [\![T']\!]_\rho$. Since $[\![T']\!]_\rho$ is a reducibility candidate, it suffices to prove $[t'/x]\sigma t \in [\![T']\!]_\rho$, since $(\lambda x.\sigma t)\ t' \rightsquigarrow [t'/x](\sigma t)$ (using call-by-name reduction). But if we let $\sigma' = \sigma[x \mapsto t']$, then we have $(\sigma', \rho) \in [\![\Gamma, x : T]\!]$ by Lemma 8, so we may apply the IH to conclude $\sigma' t \in [\![T']\!]_\rho$, as required.

Case:

$$\frac{\Gamma, X : \star \vdash t : T}{\Gamma \vdash t : \forall\, X.\, T}$$

We must prove $\sigma t \in [\![\forall\, X.\, T]\!]_\rho$. By the semantics of universal types, it suffices to assume an arbitrary $R \in \mathcal{R}$, and prove $\sigma t \in [\![T]\!]_{\rho[X \mapsto R]}$. But this follows by the IH, which we can apply because $(\sigma, \rho[X \mapsto R]) \in [\![\Gamma, X : \star]\!]$, by Lemma 8.

Case:

$$\frac{\Gamma \vdash t : \forall\, X.\, T \quad \Gamma \vdash T' : \star}{\Gamma \vdash t : [T'/X]T}$$

By the IH, we know $\sigma t \in [\![\forall\, X.\, T]\!]_\rho$, which by the semantics of universal types is equivalent to

$$\sigma t \in \bigcap_{R \in \mathcal{R}} T_{\rho[X \mapsto R]} \tag{1}$$

Since $(\sigma, \rho) \in [\![\Gamma]\!]$, we may easily observe that $\rho$ is defined for all the free type variables of $T'$. So by Lemma 7, $[\![T']\!]_\rho \in \mathcal{R}$. From the displayed formula above (1), we can conclude $\sigma t \in [\![T]\!]_{\rho[X \mapsto [\![T']\!]_\rho]}$. Now we must apply the following lemma, whose easy proof by induction on $T$ we omit, to conclude $\sigma t \in [\![[T'/X]T]\!]_\rho$.

**Lemma 10.** $[\![[T'/X]T]\!]_\rho = [\![T]\!]_{\rho[X \mapsto [\![T']\!]_\rho]}$

□

**Corollary 11.** *If $\cdot \vdash t : T$ is derivable in System F, then call-by-name reduction reaches a normal form from $t$.*

*Proof.* By Theorem 9, we have $t \in [\![T]\!]_\varnothing$. By Regularity (Theorem 5), we have $\cdot \vdash T : \star$. Then $[\![T]\!]_\varnothing$ is a reducibility candidate by Lemma 7. This means it is a subset of $\mathcal{N}$. So $t$ is call-by-name normalizing. □

A type is called **inhabited** iff there is some term that can be assigned that type in the empty context.

**Corollary 12** (Logical soundness)**.** *There is an uninhabited type.*

*Proof.* One such type is $\forall\, X.\, X$. Suppose that there is some term $t$ with $\cdot \vdash t : \forall\, X.\, X$. By Theorem 9, we have $t \in [\![\forall\, X.\, X]\!]_\varnothing$. But the interpretation of $\forall\, X.\, X$ is the intersection of all reducibility candidates. Now $\varnothing$ is a reducibility candidate (according to the definition at the start of Section 4.3). This is because it is a subset of $\mathcal{N}$ and is trivially expansion-closed. But then $t \in \varnothing$, which is impossible. □

# References


Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Commun. ACM*, 22(8): 465–476, 1979. doi:10.1145/359138.359142. URL `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.404.2715&rep=rep1&type=pdf`.

J.-Y. Girard. Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur, 1972. URL `https://www.cs.cmu.edu/afs/cs.cmu.edu/user/kw/www/scans/girard72thesis.pdf`.

J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and types*. New York, NY, USA, 1989. URL `https://www.paultaylor.eu/stable/prot.pdf`.

Brian Rabern. The history of the use of [[.]]-notation in natural language semantics. *Semantics and Pragmatics*, 9(12), 2016. doi:10.3765/sp.9.12.

W. W. Tait. Intensional interpretations of functionals of finite type i. *The Journal of Symbolic Logic*, 32(2): 198–212, 1967. doi:10.2307/2271658. URL `http://www.jstor.org/stable/2271658`.


# A Proof of type preservation for STLC (Theorem 1)

The proof is by induction on the structure of the derivation of $t \rightsquigarrow t'$.

**Case:**

$$\frac{t_1 \;\rightsquigarrow\; t_1'}{(t_1\ t_2) \;\rightsquigarrow\; (t_1'\ t_2)}\ \textit{app1}$$

In this case, the $t$ mentioned in the theorem has been refined to an application, $t_1\ t_2$. There is only one typing rule that can type an application, and so the typing derivation we are assuming we have must end with this inference:

$$\frac{\Gamma \vdash t_1 : T_2 \to T_1 \quad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1\ t_2 : T_1}$$

We may apply our induction hypothesis to the proof of $t_1 \;\rightsquigarrow\; t_1'$ and the proof in the first premise of this inference, to get:

$$\Gamma \vdash t_1' : T_2 \to T_1$$

Putting this together with our proof of $\Gamma \vdash t_2 : T_2$ (from the second premise of the typing proof above), we have

$$\frac{\Gamma \vdash t_1' : T_2 \to T_1 \quad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1'\ t_2 : T_1}$$

The case for the other application rule is similar, so we omit the details.

**Case:**

$$\frac{t \ \rightsquigarrow \ t'}{\lambda x.t \ \rightsquigarrow \ \lambda x.t'}\ lam$$

Here, we have refined the $t$ from the statement of the theorem to $\lambda x.t$. Since there is only one typing rule that can type a $\lambda$-abstraction, we know that our assumed typing derviation must end in an inference of this form:

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \lambda x.t : T_1 \to T_2}$$

Then by our induction hypothesis applied to the derivation in the premise of the reduction inference and the derivation in the premise of the typing inference, we obtain:

$$\Gamma, x : T_1 \vdash t' : T_2$$

Now we may apply the rule for typing $\lambda$-abstractions get:

$$\frac{\Gamma, x : T_1 \vdash t' : T_2}{\Gamma \vdash \lambda x.t' : T_1 \to T_2}$$

**Case:**

$$\frac{}{(\lambda x.t)\ t' \ \rightsquigarrow \ [t'/x]t}\ \beta$$

By similar reasoning about the possible form of the typing derivation, we may deduce it ends in this form:

$$\frac{\dfrac{\Gamma, x : T_2 \vdash t : T_1}{\Gamma \vdash (\lambda x.t) : T_2 \to T_1} \quad \Gamma \vdash t' : T_2}{\Gamma \vdash (\lambda x.t)\ t' : T_1}$$

To complete this case, it suffices to apply Lemma 2 to the premises of the above derivation:

$$\frac{\Gamma, x : T_2 \vdash t : T_1 \quad \Gamma \vdash t' : T_2}{\Gamma, \vdash [t'/x]t : T_1}\ \textit{Lemma 2}$$

# B Proofs of Weakening and Substitution Lemmas

We will prove Lemma 2 in the following generalized form:

**Lemma 13** (Substitution)**.** *If* $\Gamma_1, y : T_b, \Gamma_2 \vdash t_a : T_a$ *and* $\Gamma_1 \vdash t_b : T_b$*, then* $\Gamma_1, \Gamma_2 \vdash [t_b/y]t_a : T_a$*.*

The proof relies on the following lemma, which we prove first.

**Lemma 14** (Weakening)**.** *If* $\Gamma_1, \Gamma_3 \vdash t_a : T_a$ *then* $\Gamma_1, \Gamma_2, \Gamma_3 \vdash t_a : T_a$*, assuming that the variables declared in* $\Gamma_2$ *are disjoint from those declared in* $\Gamma_1$ *and* $\Gamma_3$*.*

*Proof.* The proof is by induction on the structure of the assumed derivation.

**Case:**

$$\frac{(\Gamma_1, \Gamma_3)(x) = T}{\Gamma_1, \Gamma_3 \vdash x : T}$$

We can use this inference:

$$\frac{(\Gamma_1, \Gamma_2\Gamma_3)(x) = T}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash x : T}$$

**Case:**

$$\frac{\Gamma_1, \Gamma_3, x : T_1 \vdash t : T_2}{\Gamma_1, \Gamma_3 \vdash \lambda\, x.\, t : T_1 \to T_2}$$

We can use this derivation, where we are writing (as in Chapter **??**) applications of the induction hypothesis *IH* as inferences in a derivation:

$$\frac{\dfrac{\Gamma_1, \Gamma_3, x : T_1 \vdash t : T_2}{\Gamma_1, \Gamma_2, \Gamma_3, x : T_1 \vdash t : T_2}\ IH}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \lambda\, x.\, t : T_1 \to T_2}$$

**Case:**

$$\frac{\Gamma_1, \Gamma_3 \vdash t_1 : T_2 \to T_1 \quad \Gamma_1, \Gamma_3 \vdash t_2 : T_2}{\Gamma_1, \Gamma_3 \vdash t_1\ t_2 : T_1}$$

We can use this derivation:

$$\frac{\dfrac{\Gamma_1, \Gamma_3 \vdash t_1 : T_2 \to T_1}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash t_1 : T_2 \to T_1}\ IH \quad \dfrac{\Gamma_1, \Gamma_3 \vdash t_2 : T_2}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash t_2 : T_2}\ IH}{\Gamma_1, \Gamma_3 \vdash t_1\ t_2 : T_1}$$

□

Now we can prove the Substitution Lemma:

*Proof of Lemma 13 (Substitution).* The proof is by induction on the structure of the first assumed derivation.

**Case:**

$$\frac{\Gamma(x) = T_a}{\Gamma_1, y : T_b, \Gamma_2 \vdash x : T_a}$$

Here, $t_a = x$. We must case split on whether or not $x = y$. If so, then $[t_b/y]t_a = [t_b/y]y = t_b$, and $T_a = T_b$. We construct this derivation, where we are applying Lemma 14 (Weakening) as part of the derivation:

$$\frac{\Gamma_1 \vdash t_b : T_b}{\Gamma_1, \Gamma_2 \vdash t_b : T_b}\ \textit{Lemma 14}$$

If $x \neq y$, then we use the following derivation, where we know $x$ is declared in $\Gamma_1, \Gamma_2$ since it is declared in $\Gamma = \Gamma_1, y : T_2, \Gamma_2$ and $x \neq y$:

$$\frac{(\Gamma_1, \Gamma_2)(x) = T_a}{\Gamma_1, \Gamma_2 \vdash x : T_a}$$

**Case:**

$$\frac{\Gamma_1, y : T_b, \Gamma_2, x : T_1 \vdash t : T_2}{\Gamma_1, y : T_b, \Gamma_2 \vdash \lambda\, x.\, t : T_1 \to T_2}$$

We construct this derivation, where we may assume $x \neq y$, and so the term in the conclusion, $\lambda\, x.\, [t_b/y]t$, equals the desired term $[t_b/y]\lambda\, x.\, t$:

$$\dfrac{\dfrac{\Gamma_1, y : T_b, \Gamma_2, x : T_1 \vdash t : T_2}{\Gamma_1, \Gamma_2, x : T_1 \vdash [t_b/y]t : T_2}\ IH}{\Gamma_1, \Gamma_2 \vdash \lambda\, x.\, [t_b/y]t : T_1 \to T_2}$$

**Case:**

$$\dfrac{\Gamma_1, y : T_b, \Gamma_2 \vdash t_1 : T_2 \to T_1 \quad \Gamma_1, y : T_b, \Gamma_2 \vdash t_2 : T_2}{\Gamma_1, y : T_b, \Gamma_2 \vdash t_1\ t_2 : T_1}$$

We construct this derivation, where the term in the conclusion equals the desired $[t_b/y](t_1\ t_2)$:

$$\dfrac{\dfrac{\Gamma_1, y : T_b, \Gamma_2 \vdash t_1 : T_2 \to T_1}{\Gamma_1, \Gamma_2 \vdash [t_b/y]t_1 : T_2 \to T_1} \quad \dfrac{\Gamma_1, y : T_b, \Gamma_2 \vdash t_2 : T_2}{\Gamma_1, \Gamma_2 \vdash [t_b/y]t_2 : T_2}}{\Gamma_1, \Gamma_2 \vdash [t_b/y]t_1\ [t_b/y]t_2 : T_1}$$

□