Strong Normalization for Lambda Calculi

CS252r Spring 2021: Final Project

Pratap Singh (advised by Anitha Gollamudi)

May 15, 2021

1 Introduction

This technical report aims to prove strong normalization for various calculi in the lambda cube. In class, we proved strong normalization for the simply-typed lambda calculus using the method of *logical relations*. More complex calculi—such as System F, System F_ω, LF, and the calculus of constructions—require correspondingly more complex proof techniques. We seek to understand these proof techniques as well as the practicalities and difficulties that arise when instantiating them for particular calculi.

We investigate two main proof techniques in this report. The first is the method of *candidates*, first introduced by Girard, which generalizes the method of logical relations. Instead of giving an explicit constructive definition of a logical predicate, we parameterize our logical predicate over a set of candidates—terms which satisfy the properties we require for the proof of strong normalization. We then prove that the sets of terms satisfying the logical predicate are themselves candidates, meaning that they have the requisite properties for the proof to go through. This construction avoids problems with circular definitions which can arise when types can depend on types, as described in Section 3.1. The second proof technique is by *reduction-preserving translation* to a calculus known to be strongly normalizing. We define a translation from the source calculus to a target calculus, such that if a term takes one β-reduction step in the source calculus, its translation takes at least one β-reduction step in the target calculus. Then, strong normalization of the source follows from strong normalization of the target calculus, since there can be no infinite sequence of β-reductions from a well-typed target term.

The structure of this report is as follows. Section 2 provides background by recapitulating the proof of strong normalization for the simply-typed lambda calculus. Section 3 gives in full detail the proof of strong normalization for System F, using the method of candidates. Section 4 sketches the proof of strong normalization for System F_ω, which proceeds by small modifications to the method of candidates proof from section 3. Section 5 gives in full detail the proof of strong normalization for LF, which uses a reduction-preserving translation from LF to the STLC. Finally, Section 6 discusses proving strong normalization of the calculus of constructions, which can be done by both an adaptation of the method of candidates and by a reduction-preserving translation to System F_ω. Our proofs are generally inspired by those of Sørensen [1], although we depart significantly from their syntax in favor of a more familiar presentation.

The main contributions of this work are explicit proofs of strong normalization for two complex calculi from the lambda cube, System F and LF, as well as a more general understanding of the ways in which simple proofs and proof techniques for strong normalization can be modified or extended to account for richer features.

2 Simply-Typed Lambda Calculus (STLC)

The simply-typed lambda calculus is, as the name suggests, the simplest of all the calculi in the lambda cube. We reproduce the key lemmas of the proof of strong normalization for the STLC here. For the syntax and typing rules of the STLC, as well as a full treatment of this proof, see the CS252 lecture notes.

First, we define our logical predicates, indexed by types. For STLC, we use the explicit base type of **Unit**.

Definition 1 (Logical Predicates for STLC)**.** Let SN be the set of strongly normalizing STLC terms.

$$\begin{aligned}
[\![\textsf{Unit}]\!] &\triangleq \{e \mid \quad \vdash e : \textsf{Unit} \text{ and } e \in SN\} \\
[\![\tau_1 \to \tau_2]\!] &\triangleq \{e \mid \quad \vdash e : \tau_1 \to \tau_2 \text{ and } e \in SN \text{ and } \forall e' \in [\![\tau_1]\!], e\ e' \in [\![\tau_2]\!]\}
\end{aligned}$$

(In class, we used the syntax $R_\tau(e)$ to mean $e \in [\![\tau]\!]$, but we adopt the $[\![\cdot]\!]$ syntax for ease of comparison with later sections.)

We then prove a helper lemma stating that membership in $[\![\cdot]\!]$ is preserved over β-reduction steps.

Lemma 1. *$\forall e, e', \tau$, if $\vdash e : \tau$ and $e \to e'$, then $e \in [\![\tau]\!] \iff e' \in [\![\tau]\!]$.*

Proof. Both directions proceed by induction on τ. □

This then allows us to prove the *fundamental lemma* of the logical relations proof.

Lemma 2 (Fundamental lemma)**.** *If $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e : \tau$, $v_1, \ldots, v_n$ are closed values of type $\tau_1, \ldots, \tau_n$ s.t. $\forall i \in 1..n, v_i \in [\![\tau_i]\!]$, then $e[x_1 := v_1] \ldots [x_n := v_n] \in [\![\tau]\!]$.*

Proof. Induction on the derivation of $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e : \tau$. □

Strong normalization then follows directly from Lemma 2.

Theorem 1 (Strong normalization)**.** *If $\vdash e : \tau$, then $e \in SN$.*

Proof. By specializing Lemma 2 to the empty typing context, we have $e \in [\![\tau]\!]$. But in all cases of Definition 1, we have $e \in [\![\tau]\!] \implies e \in SN$. Therefore, $e \in SN$. □

3 System F

System F extends the simply-typed lambda calculus with *polymorphism*, whereby terms are allowed to depend on types. We add universal types $\forall X.\tau$ to our grammar of types. Figure 1 gives the syntax of System F, and Figure 2 gives the typing rules.

$$\begin{aligned}
e &::= x \mid (e\ e) \mid \lambda x : \tau.\, e \mid \Lambda X.\, e \mid e\ \tau \\
\tau &::= X \mid \tau \to \tau \mid \forall X.\tau
\end{aligned}$$

Figure 1: System F grammar. Note that we avoid explicit base types, instead allowing free variables in the context which can represent base types.

$$\textsc{Var}\ \frac{}{\Gamma \vdash x : \tau}\ x : \tau \in \Gamma \qquad \textsc{Abs}\ \frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda x : \tau.\, e : \tau \to \tau'} \qquad \textsc{App}\ \frac{\Gamma \vdash e_1 : \tau \to \tau' \quad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash e_1\ e_2 : \tau'}$$

$$\textsc{Gen}\ \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \Lambda X.\, e : \forall X.\tau} \qquad \textsc{Inst}\ \frac{\Gamma \vdash e : \forall X.\tau}{\Gamma \vdash e\ \tau' : \tau[X := \tau']}$$

Figure 2: System F typing rules.

3.1 Why don't logical relations work?

Our initial approach might simply be to add a case to Definition 1 for the universal type, and modify the rest of the proof accordingly. Indeed, this approach is sufficient for many calculi, including the system of intersection types [1]. However, we run into issues when attempting to define a logical predicate for the universal type.

In particular, the correct choice of logical predicate for $\forall X.\tau$ would be as follows:

$$[\![\forall X.\tau]\!] \triangleq \bigcap \{[\![\tau[X := \tau']]\!] \mid \text{all types } \tau'\}$$

However, this definition is circular. One of the types in "all types τ'" is the very type $\forall X.\tau$ whose logical predicate we are attempting to define. Therefore, we cannot give an explicit characterization of the logical predicates for System F.

3.2 Proof by method of candidates

We cannot explicitly define the logical predicate, so instead we will proceed with any set that satisfies the properties we require for the strong normalization proof. This is the *method of candidates*, introduced by Girard.

We follow Sørensen's [1] presentation, changing notation slightly. Let SN be the set of strongly normalizing terms.

Definition 2 (Candidate)**.** A set C of System F terms is a *candidate* iff:

1. $C \subseteq SN$
2. If $e_1, \ldots, e_n \in SN$ then $x\ e_1\ \ldots\ e_n \in C$ (where x is a variable)
3. If $e_0[x := e_1]\ e_2\ \ldots\ e_n \in C$, and $e_1, \ldots, e_n \in SN$, then $(\lambda x.\, e_0)\ e_1\ e_2\ \ldots\ e_n\ \in\ C$.

Let $\mathbb{C}$ be the set of all candidates.

Definition 2 contains the properties required to show that every well-typed term is strongly normalizing. When we define the $[\![\cdot]\!]$ sets in Definition 4, we'll need to prove that they satisfy these properties in order for the subsequent strong normalization proof to work.

Definition 3 (Valuation in $\mathbb{C}$)**.** A *valuation* in $\mathbb{C}$ is a map $\xi : TV \cup B \mapsto \mathbb{C}$, where TV is the set of type variables and B is the set of base types.

Definition 4 ($[\![\cdot]\!]$)**.** For all valuations ξ and types τ, define the set $[\![\tau]\!]_\xi$ as follows:

$$\begin{array}{lcl} [\![X]\!]_\xi & \triangleq & \xi(X) \\ [\![\tau_1 \to \tau_2]\!]_\xi & \triangleq & \{e \mid e' \in [\![\tau_1]\!]_\xi \implies e\ e' \in [\![\tau_2]\!]_\xi\} \\ [\![\forall X.\tau]\!]_\xi & \triangleq & \bigcap_{C \in \mathbb{C}} [\![\tau]\!]_{\xi[X \mapsto C]} \end{array}$$

Intuitively, $[\![\tau]\!]_\xi$ is the set of strongly normalizing terms which correspond to the type τ, where the base types and type variables are mapped by ξ. The definition of $[\![\tau_1 \to \tau_2]\!]_\xi$ corresponds to the "logical" part - it strengthens the inductive hypothesis to allow us to reason across function application. In the definition of $[\![\forall X.\tau]\!]_\xi$, we avoid the circularity problems mentioned in Section 3.1 by allowing the type variable X to range over all candidates. Since candidates are defined separately to the $[\![\cdot]\!]_\xi$ sets, the recursive definition is well-founded.

Definition 5 (Computable term)**.** A term e is *computable* iff $\exists \tau, \xi$ s.t. $e \in [\![\tau]\!]_\xi$.

First, we give some helper lemmas.

Lemma 3. *$\forall$ types τ, valuations ξ, ξ', if $\forall X \in FV(\tau), \xi(X) = \xi'(X)$, then $[\![\tau]\!]_\xi = [\![\tau]\!]_{\xi'}$.*

Proof. Induction on τ.

Case X: note that X is a free variable of X, so by assumption $\xi(X) = \xi'(X)$. Since $[\![X]\!]_\xi = \xi(X) = \xi'(X)$, we have $[\![X]\!]_\xi = [\![X]\!]_{\xi'}$.

Case $\tau_1 \to \tau_2$: Our inductive hypotheses are $[\![\tau_1]\!]_\xi = [\![\tau_1]\!]_{\xi'}$ and $[\![\tau_2]\!]_\xi = [\![\tau_2]\!]_{\xi'}$.
Let $e \in [\![\tau_1 \to \tau_2]\!]_\xi$. Then, $\forall e' \in [\![\tau_1]\!]_\xi$, we have $e\ e' \in [\![\tau_2]\!]_\xi$.
Let $e' \in [\![\tau_1]\!]_\xi$. Then, by the inductive hypothesis on τ_1, $e' \in [\![\tau_1]\!]_{\xi'}$.
Likewise, by the inductive hypothesis on τ_2, we have $e\ e' \in [\![\tau_2]\!]_{\xi'}$.
Therefore, we have $e \in [\![\tau_1 \to \tau_2]\!]_{\xi'}$. So $[\![\tau_1 \to \tau_2]\!]_\xi \subseteq [\![\tau_1 \to \tau_2]\!]_{\xi'}$.
By a symmetrical argument, $[\![\tau_1 \to \tau_2]\!]_{\xi'} \subseteq [\![\tau_1 \to \tau_2]\!]_\xi$. Therefore, $[\![\tau_1 \to \tau_2]\!]_\xi = [\![\tau_1 \to \tau_2]\!]_{\xi'}$.

Case $\forall X.\tau$: Our inductive hypothesis is $\forall C \in \mathbb{C}, [\![\tau]\!]_{\xi[X\mapsto C]} = [\![\tau]\!]_{\xi'[X\mapsto C]}$.
Consider $\bigcap_{C\in\mathbb{C}} [\![\tau]\!]_{\xi[X\mapsto C]}$. By the inductive hypothesis, we can replace each of the $[\![\tau]\!]_{\xi[X\mapsto C]}$ with $[\![\tau]\!]_{\xi'[X\mapsto C]}$.
So we have $\bigcap_{C\in\mathbb{C}} [\![\tau]\!]_{\xi[X\mapsto C]} = \bigcap_{C\in\mathbb{C}} [\![\tau]\!]_{\xi'[X\mapsto C]}$, as required.

□

Lemma 4. *$\forall$ types τ, τ', type variables X, valuations ξ,*

$$[\![\tau[X := \tau']]\!]_\xi = [\![\tau]\!]_{\xi[X\mapsto[\![\tau']\!]_\xi]}.$$

Lemma 4 allows us to simplify substitutions in types by moving them into the valuation.

Proof. Induction on τ.

Case X: We must show that $[\![X[Y := \tau']]\!]_\xi = [\![X]\!]_{\xi[Y\mapsto[\![\tau']\!]_\xi]}$.
First, suppose that $X = Y$. Then, the LHS becomes $[\![\tau']\!]_\xi$, and the RHS becomes $\xi[X \mapsto [\![\tau']\!]_\xi](X) = [\![\tau']\!]_\xi$, so we are done.
Now, suppose that $X \neq Y$. Note that $X[Y := \tau'] = X$, so $[\![X[Y := \tau']]\!]_\xi = [\![X]\!]_\xi = \xi(X)$.
Furthermore, $[\![X]\!]_{\xi[Y\mapsto[\![\tau']\!]_\xi]} = \xi[Y \mapsto [\![\tau']\!]_\xi](X) = \xi(X)$, so we are done.

Case $\tau_1 \to \tau_2$: We must show that $[\![(\tau_1 \to \tau_2)[X := \tau']]\!]_\xi = [\![\tau_1 \to \tau_2]\!]_{\xi[X\mapsto[\![\tau']\!]_\xi]}$.
The inductive hypotheses are that $[\![\tau_1[X := \tau']]\!]_\xi = [\![\tau_1]\!]_{\xi[X\mapsto[\![\tau']\!]_\xi]}$ and $[\![\tau_2[X := \tau']]\!]_\xi = [\![\tau_2]\!]_{\xi[X\mapsto[\![\tau']\!]_\xi]}$.
We have:

$$\begin{aligned}
[\![(\tau_1 \to \tau_2)[X := \tau']]\!]_\xi &= [\![\tau_1[X := \tau'] \to \tau_2[X := \tau']]\!]_\xi \\
&= \{e \mid e' \in [\![\tau_1[X := \tau']]\!]_\xi \implies e\ e' \in [\![\tau_2[X := \tau']]\!]_\xi\} \\
&= \left\{e \mid e' \in [\![\tau_1]\!]_{\xi[X\mapsto[\![\tau']\!]_\xi]} \implies e\ e' \in [\![\tau_2]\!]_{\xi[X\mapsto[\![\tau']\!]_\xi]}\right\} && \text{by the IH} \\
&= [\![\tau_1 \to \tau_2]\!]_{\xi[X\mapsto[\![\tau']\!]_\xi]}
\end{aligned}$$

as required.

Case $\forall X.\tau$: This proceeds similarly to the $\tau_1 \to \tau_2$ case—we use the inductive hypothesis to replace each of the $[\![\tau[Y := \tau']]\!]_{\xi[X\mapsto C]}$ with $[\![\tau]\!]_{\xi[X\mapsto C][Y\mapsto[\![\tau']\!]_{\xi[X\mapsto C]}]}$, giving us the correct intersection. □

Lemma 5. *$SN \in \mathbb{C}$.*

Lemma 5 shows that $\mathbb{C}$ is non-empty.

Proof. We show each of the three conditions of Definition 2:

1. Show that $SN \subseteq SN$. Trivial.
2. Show that if $e_1, \ldots, e_n \in SN$ then $x\ e_1\ \ldots\ e_n \in SN$.

   Each of the e_1 reduces to a value in a finite number of steps, so we are done.

3. Show that if $e_0[x := e_1]\ e_2\ \dots\ e_n \in SN$, and $e_1, \dots, e_n \in SN$, then $(\lambda x.\, e_0)\ e_1\ e_2\ \dots\ e_n \ \in\ SN$.
   Note that the term $(\lambda x.\, e_0)\ e_1\ e_2\ \dots\ e_n$ β-reduces to $e_0[x := e_1]\ e_2\ \dots\ e_n$ in one step.
   So if $e_0[x := e_1]\ e_2\ \dots\ e_n$ terminates then $(\lambda x.\, e_0)\ e_1\ e_2\ \dots\ e_n$ must also terminate.
   So $(\lambda x.\, e_0)\ e_1\ e_2\ \dots\ e_n \in SN$, as required.

□

Lemma 6. *$\forall$ types τ, valuations ξ, $[\![\tau]\!]_\xi \in \mathbb{C}$.*

Lemma 6 shows that each of the $[\![\cdot]\!]_\xi$ sets satisfies the properties of a candidate.

Proof. Induction on τ.

Case X: note that the codomain of ξ is $\mathbb{C}$. Therefore $[\![X]\!]_\xi = \xi(X) \in \mathbb{C}$, as required.

Case $\tau_1 \to \tau_2$: Our inductive hypotheses are that $[\![\tau_1]\!]_\xi \in \mathbb{C}$ and $[\![\tau_2]\!]_\xi \in \mathbb{C}$.
Note that $[\![\tau_1 \to \tau_2]\!]_\xi = \{e \mid e' \in [\![\tau_1]\!]_\xi \implies e\ e' \in [\![\tau_2]\!]_\xi\}$.
For ease of reading, let $S = \{e \mid e' \in [\![\tau_1]\!]_\xi \implies e\ e' \in [\![\tau_2]\!]_\xi\}$. We show the three conditions of definition 2:

1. Show that $S \subseteq SN$.
   Let $e \in S$, $e' \in [\![\tau_1]\!]_\xi$, so that $e\ e' \in [\![\tau_2]\!]_\xi$. By the second inductive hypothesis, we know that $e\ e' \in SN$, so the evaluation of $e\ e'$ must terminate for any choice of e'. Therefore, e must also terminate, i.e. $e \in SN$. Therefore $S \subseteq SN$.

2. Show that if $e_1, \dots, e_n \in SN$ then $x\ e_1\ \dots\ e_n \in S$.
   Let $e' \in [\![\tau_1]\!]_\xi$, s.t. we must show that $x\ e_1\ \dots\ e_n\ e' \in [\![\tau_2]\!]_\xi$. But by the first inductive hypothesis, we know that $e' \in SN$. So we can use condition 2 of the second inductive hypothesis to conclude $x\ e_1\ \dots\ e_n\ e' \in [\![\tau_2]\!]_\xi$, as required.

3. Show that if $e_0[x := e_1]\ e_2\ \dots\ e_n \in S$, and $e_1, \dots, e_n \in SN$, then $(\lambda x.\, e_0)\ e_1\ e_2\ \dots\ e_n \ \in\ S$.
   Again, let $e' \in [\![\tau_1]\!]_\xi$, s.t. we must show that $(\lambda x.\, e_0)\ e_1\ e_2\ \dots\ e_n\ e' \ \in\ [\![\tau_2]\!]_\xi$.
   By the first inductive hypothesis, we know that $e' \in SN$. So we can use condition 3 of the second inductive hypothesis to conclude $(\lambda x.\, e_0)\ e_1\ e_2\ \dots\ e_n\ e' \ \in\ [\![\tau_2]\!]_\xi$, as required.

Case $\forall X.\tau$: Our inductive hypothesis is that $\forall C \in \mathbb{C}, [\![\tau]\!]_{\xi[X \mapsto C]} \in \mathbb{C}$.

We first prove that the intersection of two candidates is a candidate.
Suppose we have candidates $C_1, C_2 \in \mathbb{C}$. Now, $C_1 \subseteq SN$ and $C_2 \subseteq SN$, so $C_1 \cap C_2 \subseteq SN$.
Now, suppose $e_1, \dots, e_n \in SN$. We have $x\ e_1\ \dots\ e_n \in C_1$ and $x\ e_1\ \dots\ e_n \in C_2$, so $x\ e_1\ \dots\ e_n \in C_1 \cap C_2$.
Similarly, suppose $e_0[x := e_1]\ e_2\ \dots\ e_n \in C_1$ and $e_0[x := e_1]\ e_2\ \dots\ e_n \in C_2$. Then we have $(\lambda x.\, e_0)\ e_1\ e_2\ \dots\ e_n \ \in\ C_1$ and $(\lambda x.\, e_0)\ e_1\ e_2\ \dots\ e_n \ \in\ C_2$, so $(\lambda x.\, e_0)\ e_1\ e_2\ \dots\ e_n \ \in\ C_1 \cap C_2$.
That is, $C_1 \cap C_2$ satisfies all the properties of a candidate. So the intersection of two candidates is a candidate.
By trivial induction, the intersection of any finite number of candidates is a candidate as well.

Returning to the $\forall X.\tau$ case, note that $[\![\tau]\!]_{\xi[X \mapsto C]} = \bigcap_{C \in \mathbb{C}} [\![\tau]\!]_{\xi[X \mapsto C]}$. By the inductive hypothesis, each of the $[\![\tau]\!]_{\xi[X \mapsto C]}$ is a candidate. Since the intersection of candidates is a candidate, we have that $\bigcap_{C \in \mathbb{C}} [\![\tau]\!]_{\xi[X \mapsto C]} \in \mathbb{C}$, as required. □

With the helper lemmas out of the way, we can give the main proof of strong normalization.

Definition 6 (Computable instance of a term)**.** For term e, let $\Gamma \vdash e : \tau$, where $\mathrm{dom}(\Gamma) = \{x_1, \dots, x_k\}$. Then for some valuation ξ, let $e_i \in [\![\Gamma(x_i)]\!]_\xi$ for $i \in 1..k$. Then the term $e[x_1 := e_1][x_2 := e_2] \dots [x_k := e_k]$ is a *computable instance* of e.

Intuitively, a computable instance of a (possibly open) term is a closed term of the same type, where each of the free variables is substituted by a term drawn from the candidate set for that type.

Lemma 7. *A computable instance of a well-typed term is computable.*

This is the *fundamental lemma* of this proof, corresponding to the fundamental lemma of a logical relations proof. As before, we use the computable instance definition to consider open terms, strengthening the inductive hypothesis to allow the application case to go through.

Proof. Let e be a well-typed term s.t. $\Gamma \vdash e : \tau$, where $\Gamma = x_1 : \tau_1, \ldots, x_k : \tau_k$. Let ξ be a valuation, and $e_i \in [\![\Gamma(x_i)]\!]_\xi$ for $i \in 1..k$. Let $c_e = e[x_1 := e_1][x_2 := e_2] \ldots [x_k := e_k]$ be a computable instance of e. We show that c_e is in some candidate. We proceed by induction on the derivation of $\Gamma \vdash e : \tau$.

Case Var: we have $e = x_i$ and $\tau = \tau_i$ for some $i \in 1..k$. So $e[x_1 := e_1] \ldots [x_k := e_k] = e_i$. By assumption, $e_i \in [\![\tau_i]\!]_\xi$, as required.

Case Abs: we have $e = \lambda x.\, e'$, $\tau = \tau_x \to \tau'$, and $\Gamma, x : \tau_x \vdash e' : \tau'$.
The IH is that $\forall \xi$, if $e_i \in [\![\tau_i]\!]_\xi$ for $i \in 1..k$ and $e_x \in [\![\tau_x]\!]_\xi$, then $e'[x_1 := e_1] \ldots [x_k := e_k][x := e_x] \in [\![\tau']\!]_\xi$.
We will show that $c_e = \lambda x.\, e'[x_1 := e_1] \ldots [x_k := e_k] \in [\![\tau_x \to \tau']\!]_\xi$.
Let $e_x \in [\![\tau_x]\!]_\xi$, s.t. we need to show $c_e\ e_x \in [\![\tau']\!]_\xi$. Note that since $e_x \in [\![\tau_x]\!]_\xi$, we can apply the IH and conclude $e'[x_1 := e_1] \ldots [x_k := e_k][x := e_x] \in [\![\tau']\!]_\xi$.
Since $x \notin \mathrm{dom}(\Gamma)$, the substitutions commute, and we have $e'[x := e_x][x_1 := e_1] \ldots [x_k := e_k] \in [\![\tau']\!]_\xi$.
Now, by Lemma 6, $[\![\tau']\!]_\xi$ is a candidate. By property 3 of Definition 2, we can therefore conclude that $(\lambda x.\, e'\ e_x)[x_1 := e_1] \ldots [x_k := e_k] \in [\![\tau']\!]_\xi$. Again, since the substitutions commute with the application of e_x, we have $(\lambda x.\, e'[x_1 := e_1] \ldots [x_k := e_k])\ e_x \in [\![\tau']\!]_\xi$, i.e. $c_e\ e_x \in [\![\tau']\!]_\xi$, as required.

Case App: we have $e = e_a\ e'$, $\Gamma \vdash e_a : \tau' \to \tau$, and $\Gamma \vdash e' : \tau'$.
The inductive hypotheses are that $\forall \xi$, if $e_i \in [\![\tau_i]\!]_\xi$ for $i \in 1..k$, then $e_a[x_1 := e_1] \ldots [x_k := e_k] \in [\![\tau' \to \tau]\!]_\xi$ and $e'[x_1 := e_1] \ldots [x_k := e_k] \in [\![\tau']\!]_\xi$.
By definition of $[\![\tau' \to \tau]\!]_\xi$, we have $(e_a[x_1 := e_1] \ldots [x_k := e_k])\ \ (e'[x_1 := e_1] \ldots [x_k := e_k]) \in [\![\tau]\!]_\xi$.
Equivalently, $(e_a\ e')[x_1 := e_1] \ldots [x_k := e_k] \in [\![\tau]\!]_\xi$ as required.

Case Gen: we have $e = \Lambda X.\, e'$, $\tau = \forall X. \tau'$, and $\Gamma \vdash e' : \tau'$, with $X \notin FV(\Gamma)$.
The inductive hypothesis is that $\forall \xi$, if $e_i \in [\![\tau_i]\!]_\xi$ for $i \in 1..k$, then $e'[x_1 := e_1] \ldots [x_k := e_k] \in [\![\tau']\!]_\xi$.
Now, since $X \notin FV(\Gamma)$, we know that $X \notin FV(\tau_i)$ for every i. Therefore, $\forall$ candidates C, $\forall i \in 1..k$, $\forall Y \in FV(\tau_i)$, we have $\xi(Y) = \xi[X \mapsto C](Y)$. So we can apply Lemma 3 to conclude that $\forall C, \forall i$, $[\![\tau_i]\!]_\xi = [\![\tau_i]\!]_{\xi[X \mapsto C]}$.
By the inductive hypothesis (taking $\xi[X \mapsto C]$ as our valuation), we can then conclude that $e'[x_1 := e_1] \ldots [x_k := e_k] \in [\![\tau']\!]_{\xi[X \mapsto C]}$ for every candidate C, i.e. $e'[x_1 := e_1] \ldots [x_k := e_k] \in \bigcap_{C \in \mathbb{C}} [\![\tau']\!]_{\xi[X \mapsto C]}$.
Since $[\![\forall X. \tau]\!]_\xi = \bigcap_{C \in \mathbb{C}} [\![\tau]\!]_{\xi[X \mapsto C]}$, we have $e'[x_1 := e_1] \ldots [x_k := e_k] \in [\![\forall X. \tau]\!]_\xi$.
Now, note that $\Lambda X.\, (e'[x_1 := e_1] \ldots [x_k := e_k]) = (\Lambda X.\, e')[x_1 := e_1] \ldots [x_k := e_k]$, since $X \notin FV(\Gamma)$.
So we have $(\Lambda X.\, e')[x_1 := e_1] \ldots [x_k := e_k] = c_e \in [\![\forall X. \tau]\!]_\xi$, as required.

Case Inst: we have $e = e'\ \tau'$, $\Gamma \vdash e' : \forall X. \tau_X$, and $\tau = \tau_X[X := \tau']$.
The inductive hypothesis is that $\forall \xi$, if $e_i \in [\![\tau_i]\!]_\xi$ for $i \in 1..k$, then $e'[x_1 := e_1] \ldots [x_k := e_k] \in [\![\forall X. \tau_X]\!]_\xi$.
Applying the inductive hypothesis and unfolding the definition of $[\![\forall X. \tau_X]\!]_\xi$, we have that $e'[x_1 := e_1] \ldots [x_k := e_k] \in [\![\tau_X]\!]_{\xi[x \mapsto C]}$ for every candidate C.
In particular, $[\![\tau']\!]_\xi$ is a candidate by Lemma 6, so $e'[x_1 := e_1] \ldots [x_k := e_k] \in [\![\tau_X]\!]_{\xi[x \mapsto [\![\tau']\!]_\xi]}$.
But by Lemma 4, $[\![\tau_X]\!]_{\xi[x \mapsto [\![\tau']\!]_\xi]} = [\![\tau_X[X := \tau']]\!]_\xi = [\![\tau]\!]_\xi$. So $e'[x_1 := e_1] \ldots [x_k := e_k] \in [\![\tau]\!]_\xi$.
Therefore $(e'\ \tau')[x_1 := e_1] \ldots [x_k := e_k] \in [\![\tau]\!]_\xi$, as required.

□

We can now finish the proof of strong normalization.

Theorem 2 (Strong normalization)**.** *Every well-typed term in System F is strongly normalizing.*

Proof. Let e be a well-typed System F term, then $\vdash e : \tau$. Therefore, since the typing context is empty, we don't need to substitute anything for the free variables; so e is a computable instance of itself. Now, by Lemma 7, we have that e is computable, so $e \in [\![\tau]\!]_\xi$ for some ξ. Then by Lemma 6, $[\![\tau]\!]_\xi \in \mathbb{C}$. Now, by the first part of the definition of a candidate, $[\![\tau]\!]_\xi \subseteq SN$, so $e \in SN$. $\square$

4 System F$_\omega$

We now consider the calculus System F$_\omega$, which is defined by adding type constructors to System F. The syntax of System F$_\omega$ is given in Figure 3. The proof of strong normalization of System F$_\omega$ is a straightforward adaptation of that of System F. We give only a sketch of the proof here, highlighting the key differences.

$$\begin{aligned} e &::= x \mid (e\ e) \mid \lambda x : \tau.\, e \mid \Lambda X.\, e \mid e\ \tau \\ \tau &::= X \mid \tau \to \tau \mid \forall X.\tau \mid \tau\ \tau \mid \lambda X\!:\!\kappa.\, \tau \\ \kappa &::= * \mid \kappa \to \kappa \end{aligned}$$

Figure 3: System F$_\omega$ grammar.

First, we must change our definitions of candidate and valuation to allow for multiple kinds. We now have a family of candidates indexed by kinds, as follows:

Definition 7 (Candidate)**.** For every kind κ, we define a candidate set $\mathbb{C}_\kappa$. Let $\mathbb{C}_*$ be the set of all sets of terms that satisfy the three conditions of Definition 2. Let $\mathbb{C}_{\kappa_1 \to \kappa_2}$ be the set of all functions $f : \mathbb{C}_{\kappa_1} \to \mathbb{C}_{\kappa_2}$.

Valuations map type variables to candidates of their respective kinds:

Definition 8 (Valuation in $\mathbb{C}_\kappa$)**.** $\forall \kappa$, a valuation is a function $\xi : TV_\kappa \mapsto \mathbb{C}_\kappa$, where TV_κ is the set of all type variables of kind κ.

We can now define our sets $[\![\cdot]\!]_\xi$. The definition is similar to Definition 4, adding cases for the two new syntactic constructs:

Definition 9 ($[\![\cdot]\!]_\xi$)**.** For all valuations ξ and types τ, define the set $[\![\tau]\!]_\xi$ as follows:

$$\begin{array}{lcl} [\![X]\!]_\xi & \triangleq & \xi(X) \\ [\![\tau_1\ \tau_2]\!]_\xi & \triangleq & [\![\tau_1]\!]_\xi\ [\![\tau_2]\!]_\xi \\ [\![\lambda X\!:\!\kappa.\, \tau]\!]_\xi & \triangleq & C \mapsto [\![\tau]\!]_{\xi[X \mapsto C]} \\ [\![\tau_1 \to \tau_2]\!]_\xi & \triangleq & \{e \mid e' \in [\![\tau_1]\!]_\xi \implies e\ e' \in [\![\tau_2]\!]_\xi\} \\ [\![\forall X.\tau]\!]_\xi & \triangleq & \bigcap_{C \in \mathbb{C}} [\![\tau]\!]_{\xi[X \mapsto C]} \end{array}$$

where the $[\![\lambda X\!:\!\kappa.\, \tau]\!]_\xi$ case is a function from candidates of kind κ to candidates of the kind of τ.

The definition of a computable term is identical to that of Definition 5.

With these definitions, we can prove strong normalization for System F$_\omega$. Since the proof is largely a mechanical adaptation of that of Section 3.2, we give only the lemma statements and the overall proof strategy.

Lemma 8. *$\forall \kappa$, $\forall \tau$ of kind κ, $\forall \xi, \xi'$ over kind κ, if $\forall X \in FV(\tau), \xi(X) = \xi'(X)$, then $[\![\tau]\!]_\xi = [\![\tau]\!]_{\xi'}$.*

Proof. Induction on κ, with induction on τ in the base case $*$. $\square$

Lemma 9. *$\forall \kappa$, $\forall \tau, \tau'$ of kind κ, $\forall \xi$ over kind κ, $[\![\tau[X := \tau']]\!]_\xi = [\![\tau]\!]_{\xi[X \mapsto [\![\tau']\!]_\xi]}$.*

Proof. Induction on κ, with induction on τ in the base case $*$. $\square$

Lemma 10. *$SN \in \mathbb{C}_*$.*

Proof. Identical to the proof of Lemma 5. □

Lemma 11. *$\forall\kappa$, $\forall\tau$ of kind κ, $\forall\xi$, $[\![\tau]\!]_\xi \in \mathbb{C}_\kappa$.*

Proof. Induction on κ, with induction on τ in the base case $*$. □

We can now give the fundamental lemma and finish the proof:

Lemma 12 (Fundamental lemma)**.** *A computable instance of a well-typed term is computable.*

Proof. Induction on the derivation of $\Gamma \vdash e : \tau$, where e is the well-typed term. □

Theorem 3 (Strong normalization)**.** *Every well-typed term in System F_ω is strongly normalizing.*

Proof. Let e be a well-typed System F_ω term, then $\vdash e : \tau$. As in the proof of Theorem 2, e is a computable instance of itself. Now, by Lemma 12, we have that e is computable, so $e \in [\![\tau]\!]_\xi$ for some ξ over kind κ. Then by Lemma 11, $[\![\tau]\!]_\xi \in \mathbb{C}_*$. Now, by the first part of Definition 7, $[\![\tau]\!]_\xi \subseteq SN$, so $e \in SN$. □

5 LF: First Order Dependent Type System

The calculus LF extends the simply-typed lambda calculus with dependent types, which allow types to depend on terms. We also have type constructors, as with System F_ω. Figure 4 gives the syntax of LF, and Figure 5 gives the kinding and typing rules. For more details on LF, see the CS252 lecture notes.

$$\begin{aligned}
e &::= x \mid (e\ e) \mid \lambda x : \tau.\, e \\
\tau &::= X \mid \Pi x{:}\tau.\tau \mid \tau\ e \mid \lambda x{:}\tau.\,\tau \\
\kappa &::= * \mid \Pi x{:}\tau.\kappa
\end{aligned}$$

Figure 4: LF grammar.

$$\textsc{KVar}\ \frac{}{\Gamma \vdash X:\kappa}\ X:\kappa \in \Gamma \qquad \textsc{KPi}\ \frac{\Gamma \vdash \tau_1 : * \quad \Gamma, x:\tau_1 \vdash \tau_2 : *}{\Gamma \vdash \Pi x{:}\tau_1.\tau_2 : *} \qquad \textsc{KApp}\ \frac{\Gamma \vdash \tau : \Pi x{:}\tau'.\kappa \quad \Gamma \vdash e:\tau'}{\Gamma \vdash \tau\ e : \kappa[x := e]}$$

$$\textsc{KConv}\ \frac{\Gamma \vdash \tau:\kappa \quad \Gamma \vdash \kappa \equiv \kappa'}{\Gamma \vdash \tau:\kappa'} \qquad \textsc{KConstr}\ \frac{\Gamma \vdash \tau_1 : * \quad \Gamma, x:\tau_1 \vdash \tau_2 : \kappa}{\Gamma \vdash \lambda x{:}\tau_1.\,\tau_2 : \Pi x{:}\tau_1.\kappa}$$

$$\textsc{TVar}\ \frac{\Gamma \vdash \tau : *}{\Gamma \vdash x:\tau}\ x:\tau \in \Gamma \qquad \textsc{TAbs}\ \frac{\Gamma \vdash \tau : * \quad \Gamma, x:\tau \vdash e:\tau'}{\Gamma \vdash \lambda x{:}\tau.\, e : \Pi x{:}\tau.\tau'}$$

$$\textsc{TApp}\ \frac{\Gamma \vdash e_1 : \Pi x{:}\tau.\tau' \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ e_2 : \tau'[x := e_2]} \qquad \textsc{TConv}\ \frac{\Gamma \vdash e:\tau \quad \Gamma \vdash \tau \equiv \tau' : \kappa}{\Gamma \vdash e:\tau'}$$

Figure 5: LF kinding and typing rules. We omit the definitions of $\Gamma \vdash \tau_1 \equiv \tau_2 : \kappa$ and $\Gamma \vdash \kappa_1 \equiv \kappa_2$.

LF is strongly normalizing: the proof is by reduction-preserving translation to the simply typed lambda calculus [2].

Contraction Maps Certain properties of predicate logic (e.g., LF and other dependent type systems) can be studied at the propositional level (e.g., STLC). Strong normalization is one such property. We seek to define a translation from first-order to propositional formulas, and a matching translation on proofs; that is, a translation from LF to STLC that preserves the strong normalization property.

Fix an STLC base type 0_τ. Figure 6 is the first attempt at a translation from LF to STLC. Note that this is not strong enough to prove the SN. The contraction map, $\mathfrak{b}$, ignores the reductions at the type level.

$\mathfrak{b}(*)$	$= 0_\tau$	Proper kind
$\mathfrak{b}(\Pi x{:}\tau.\kappa)$	$= \mathfrak{b}(\tau) \to \mathfrak{b}(\kappa)$	First-order kind
$\mathfrak{b}(\Pi x{:}\tau_1.\tau_2)$	$= \mathfrak{b}(\tau_1) \to \mathfrak{b}(\tau_2)$	Dependent function type
$\mathfrak{b}(\lambda x{:}\tau_1.\,\tau_2)$	$= \mathfrak{b}(\tau_2)$	Type constructor
$\mathfrak{b}(X)$	$= X_X$	Constructor variable
$\mathfrak{b}(\tau\ e)$	$= \mathfrak{b}(e)$	Type application
$\mathfrak{b}(x)$	$= x$	Term variable
$\mathfrak{b}(e_1\ e_2)$	$= \mathfrak{b}(e_1)\ \mathfrak{b}(e_2)$	Term application
$\mathfrak{b}(\lambda x{:}\tau.\,e)$	$= \lambda x{:}\mathfrak{b}(\tau).\,\mathfrak{b}(e)$	Function term

Figure 6: Contracting map from LF to STLC. X is a constructor variable—has kind $\Pi x : \tau.\kappa$—and it gets translated to a type variable that has simple kind.

Translation of typing context is defined as follows.

$$\mathfrak{b}(\Gamma) = \{x : \mathfrak{b}(\tau) \mid (x : \tau) \in \Gamma\} \cup \{x_\alpha : \mathfrak{b}(\kappa) \mid (\alpha : \kappa) \in \Gamma\}$$

That is, for every constructor variable α, a term variable x_α is introduced in the translated typing context.

Lemma 13 (Type-Preserving Translation)**.** *If $\Gamma \vdash e : \tau$ then $\mathfrak{b}(\Gamma) \vdash \mathfrak{b}(e) : \mathfrak{b}(\tau)$.*

Proof. Induction on the derivation of the typing judgment. We omit the details since this lemma is not used in our eventual SN proof. □

However, the translation does not preserve the number of reductions in the source language.

Lemma 14 (Not Reduction-Preserving Translation)**.** *If $\Gamma \vdash e : \tau$ and $e \to_\beta e'$ then $\mathfrak{b}(e) \to_\beta \mathfrak{b}(e')$ or $\mathfrak{b}(e) = \mathfrak{b}(e')$. The latter case only happens when the reduction is performed withing a type contained in e.*

Lemma 14 implies that the translated terms have fewer reductions than the source term. Specifically, it does not allow us to apply induction hypothesis to type-level reductions, since they are not subterms. Thus, we cannot prove that there is no infinite sequence of reductions at the type-level.

We fix this using a different translation for types and terms that is reduction-preserving. The central idea of the new translation is that whenever a type dependency is removed, a new term corresponding to the dependent type is added. We elaborate on this idea in the next section.

5.1 Proof by reduction-preserving translation

We follow Sørensen's [1] presentation for this proof. The main translation $e^\sharp$ depends on $\mathfrak{b}$, so we first give some lemmas about $\mathfrak{b}$.

Lemma 15. $\forall \tau, e,\ \mathfrak{b}(\tau[x := e]) = \mathfrak{b}(\tau)$

Proof. Consider two cases: $x \in FV(\tau)$ and $x \notin FV(\tau)$. Note that in the second case, $\tau[x := e] = \tau$, so we are done. We prove the first case by induction on τ.

Case X: There is nothing to substitute since x is a term-level variable. We are done.

Case $\Pi y\!:\!\tau_1.\tau_2$: The inductive hypotheses are that $\forall x, e$ s.t. $x \in FV(\tau_1)$, $\mathfrak{b}(\tau_1[x := e]) = \mathfrak{b}(\tau_1)$, and $\forall x, e$ s.t. $x \in FV(\tau_2) \setminus y$, $\mathfrak{b}(\tau_2[x := e]) = \mathfrak{b}(\tau_2)$. Now, $\mathfrak{b}(\Pi y\!:\!\tau_1.\tau_2) = \mathfrak{b}(\tau_1) \to \mathfrak{b}(\tau_2)$. We have:

$$\begin{aligned}
\mathfrak{b}((\Pi y\!:\!\tau_1.\tau_2)[x := e]) &= \mathfrak{b}(\Pi y\!:\!\tau_1[x := e].\tau_2[x := e]) \\
&= \mathfrak{b}(\tau_1[x := e]) \to \mathfrak{b}(\tau_2[x := e]) \\
&= \mathfrak{b}(\tau_1) \to \mathfrak{b}(\tau_2) \qquad \text{by the inductive hypotheses} \\
&= \mathfrak{b}(\Pi y\!:\!\tau_1.\tau_2)
\end{aligned}$$

as required.

Case $\tau\ e'$: The inductive hypothesis is that $\forall x, e$ s.t. $x \in FV(\tau)$, $\mathfrak{b}(\tau[x := e]) = \mathfrak{b}(\tau)$.
We have $\mathfrak{b}((\tau\ e')[x := e]) = \mathfrak{b}((\tau[x := e]\ e'[x := e])) = \mathfrak{b}(e'[x := e])$. But x must not be a free variable of e', so $\mathfrak{b}(e'[x := e]) = \mathfrak{b}(e')$, completing the case.

Case $\lambda y\!:\!\tau_1.\,\tau_2$: The inductive hypotheses are that $\forall x, e$ s.t. $x \in FV(\tau_1)$, $\mathfrak{b}(\tau_1[x := e]) = \mathfrak{b}(\tau_1)$, and $\forall x, e$ s.t. $x \in FV(\tau_2) \setminus y$, $\mathfrak{b}(\tau_2[x := e]) = \mathfrak{b}(\tau_2)$.
Now, $\mathfrak{b}((\lambda y\!:\!\tau_1.\,\tau_2)[x := e]) = \mathfrak{b}(\lambda y\!:\!(\tau_1[x := e]).\,\tau_2[x := e]) = \mathfrak{b}(\tau_2[x := e])$.
But by the inductive hypothesis, $\mathfrak{b}(\tau_2[x := e]) = \mathfrak{b}(\tau_2)$. Note that $\mathfrak{b}(\lambda y\!:\!\tau_1.\,\tau_2) = \mathfrak{b}(\tau_2)$, so we are done.

□

Lemma 16. *$\forall \kappa, e$, $\mathfrak{b}(\kappa[x := e]) = \mathfrak{b}(\kappa)$*

Proof. Induction on κ.

Case $*$. Note that $*[x := e] = *$, so we are done.

Case $\Pi y\!:\!\tau.\kappa$. The inductive hypothesis is that $\forall x, e$, $\mathfrak{b}(\kappa[x := e]) = \mathfrak{b}(\kappa)$. Furthermore, by Lemma 15, we have $\forall x, e$ s.t. $x \in FV(\tau)$, $\mathfrak{b}(\tau[x := e]) = \mathfrak{b}(\tau)$. Now, $\mathfrak{b}(\Pi y\!:\!\tau.\kappa) = \mathfrak{b}(\tau) \to \mathfrak{b}(\kappa)$. We have:

$$\begin{aligned}
\mathfrak{b}((\Pi y\!:\!\tau.\kappa)[x := e]) &= \mathfrak{b}(\Pi y\!:\!\tau[x := e].\kappa[x := e]) \\
&= \mathfrak{b}(\tau[x := e]) \to \mathfrak{b}(\kappa[x := e]) \\
&= \mathfrak{b}(\tau) \to \mathfrak{b}(\kappa) \qquad \text{by the inductive hypothesis and Lemma 15} \\
&= \mathfrak{b}(\Pi y\!:\!\tau.\kappa)
\end{aligned}$$

as required.

□

Lemma 17. *$\forall \Gamma, \tau_1, \tau_2, \kappa$, if $\Gamma \vdash \tau_1 \equiv \tau_2 \!:\! \kappa$, then $\mathfrak{b}(\tau_1) = \mathfrak{b}(\tau_2)$*

Proof. Induction on the derivation of $\Gamma \vdash \tau_1 \equiv \tau_2 \!:\! \kappa$. The cases are straightforward, hence omitted. □

Lemma 18. *$\forall \Gamma, \kappa_1, \kappa_2$, if $\Gamma \vdash \kappa_1 \equiv \kappa_2$, then $\mathfrak{b}(\kappa_1) = \mathfrak{b}(\kappa_2)$*

Proof. Induction on the derivation of $\Gamma \vdash \kappa_1 \equiv \kappa_2$. The cases are straightforward, hence omitted. □

We can now give the definition of $e^\sharp$, the translation from LF to the STLC which preserves β-reductions and gives us the required strong normalization result. Figure 7 gives this translation. Since kinds do not appear in terms, we do not give a translation for them.

The translation of Figure 7 preserves β-reductions that occur at the type level, even though those do not have any effect on the semantics of the underlying STLC term. We will prove this in Lemma 21. Intuitively, however, note that the translations of type constructors and function terms add an extra $\lambda z : 0_\tau$ abstraction which is

$(\Pi x\!:\!\tau_1.\tau_2)^\sharp$	$= y_{\mathfrak{b}(\tau_1)}\ \tau_1^\sharp\ \left(\lambda x\!:\!\mathfrak{b}(\tau_1).\,\tau_2^\sharp\right)$	Dependent function type
$(\lambda x\!:\!\tau_1.\,\tau_2)^\sharp$	$= \left(\lambda z\!:\!0_\tau.\,\lambda x\!:\!\mathfrak{b}(\tau_1).\,\tau_2^\sharp\right)\ \tau_1^\sharp$	Type constructor
$X^\sharp$	$= x_X$	Type variable
$(\tau\ e)^\sharp$	$= \tau^\sharp\ e^\sharp$	Type application
$x^\sharp$	$= x$	Term variable
$(e_1\ e_2)^\sharp$	$= e_1^\sharp\ e_2^\sharp$	Term application
$(\lambda x\!:\!\tau.\,e)^\sharp$	$= \left(\lambda z\!:\!0_\tau.\,\lambda x\!:\!\mathfrak{b}(\tau).\,e^\sharp\right)\ \tau^\sharp$	Function term

where:

$$\begin{aligned} x_X &= \text{a fresh STLC term variable for each LF type variable } X \\ y_\tau &= \text{a fresh STLC term variable for each LF base type } \tau \end{aligned}$$

Figure 7: β-reduction-preserving translation from LF to STLC

applied to the translation of the type. This means that any type-level reductions will take place when reducing the translation of the type.

We first prove two lemmas that show that substitution distributes over our $e^\sharp$ translation.

Lemma 19. *The following two properties hold:*

1. $\forall \tau, x, e,\ (\tau[x := e])^\sharp = \tau^\sharp[x := e^\sharp]$
2. $\forall e, x, e',\ (e[x := e'])^\sharp = e^\sharp[x := e'^\sharp]$.

Proof. The proof is by simultaneous induction on τ and e, proving both properties. Note that since types and terms are defined in terms of each other, this is well-founded. Our inductive hypotheses will always refer to "smaller" component terms and types of the case we are in.

Case X: There is nothing to substitute since x is a term-level variable. We are done.

Case $\Pi y\!:\!\tau_1.\tau_2$: The inductive hypotheses are that $\forall x, e$ s.t. $x \in FV(\tau_1)$, $(\tau_1[x := e])^\sharp = \tau_1^\sharp[x := e^\sharp]$, and $\forall x, e$ s.t. $x \in FV(\tau_2) \setminus y$, $(\tau_2[x := e])^\sharp = \tau_2^\sharp[x := e^\sharp]$. Now, using the inductive hypotheses and Lemma 15:

$$\begin{aligned} ((\Pi y\!:\!\tau_1.\tau_2)[x := e])^\sharp &= (\Pi y\!:\!\tau_1[x := e].\tau_2[x := e])^\sharp \\ &= y_{\mathfrak{b}(\tau_1[x:=e])}\ (\tau_1[x := e])^\sharp\ \left(\lambda x\!:\!\mathfrak{b}(\tau_1[x := e]).\,(\tau_2[x := e])^\sharp\right) \\ &= y_{\mathfrak{b}(\tau_1)}\ (\tau_1^\sharp[x := e^\sharp])\ (\lambda x\!:\!\mathfrak{b}(\tau_1).\,\tau_2^\sharp[x := e^\sharp]) \\ &= \left(y_{\mathfrak{b}(\tau_1)}\ \tau_1^\sharp\ (\lambda x\!:\!\mathfrak{b}(\tau_1).\,\tau_2^\sharp)\right)[x := e^\sharp] \\ &= (\Pi y\!:\!\tau_1.\tau_2)^\sharp[x := e^\sharp] \end{aligned}$$

as required.

Case $\tau\ e$: The inductive hypotheses are that $\forall x, e'$ s.t. $x \in FV(\tau)$, $(\tau[x := e'])^\sharp = \tau^\sharp[x := e'^\sharp]$, and $\forall x, e'$ s.t. $x \in FV(e)$, $(e[x := e'])^\sharp = e^\sharp[x := e'^\sharp]$. Using the inductive hypotheses, we have:

$$\begin{aligned} (\tau\ e)[x := e']^\sharp &= (\tau[x := e']\ e[x := e'])^\sharp \\ &= (\tau[x := e'])^\sharp\ (e[x := e'])^\sharp \\ &= (\tau^\sharp[x := e'^\sharp])\ (e^\sharp[x := e'^\sharp]) \\ &= (\tau^\sharp\ e^\sharp)[x := e'^\sharp] \\ &= (\tau\ e)^\sharp[x := e'^\sharp] \end{aligned}$$

as required.

Case $\lambda y\!:\!\tau_1.\,\tau_2$**:** The inductive hypotheses are that $\forall x, e'$ s.t. $x \in FV(\tau_1)$, $(\tau_1[x := e'])^\sharp = \tau_1^\sharp[x := e'^\sharp]$ and $\forall x, e'$ s.t. $x \in FV(\tau_2) \setminus y$, $(\tau_2[x := e'])^\sharp = \tau_2^\sharp[x := e'^\sharp]$. Now, using the inductive hypotheses and Lemma 15:

$$\begin{aligned}
((\lambda y\!:\!\tau_1.\,\tau_2)[x := e'])^\sharp &= (\lambda y\!:\!\tau_1[x := e'].\,\tau_2[x := e'])^\sharp \\
&= \left(\lambda z\!:\!0_\tau.\,\lambda x\!:\!\mathfrak{b}(\tau_1[x := e']).\,(\tau_2[x := e'])^\sharp\right)\ (\tau_1[x := e'])^\sharp \\
&= \Big(\lambda z\!:\!0_\tau.\,\lambda x\!:\!\mathfrak{b}(\tau_1).\,(\tau_2^\sharp[x := e'^\sharp])\Big)\ \tau_1^\sharp[x := e'^\sharp] \\
&= \Big(\Big(\lambda z\!:\!0_\tau.\,\lambda x\!:\!\mathfrak{b}(\tau_1).\,\tau_2^\sharp\Big)\ \tau_1^\sharp\Big)\,[x := e'^\sharp] \\
&= (\lambda y\!:\!\tau_1.\,\tau_2)^\sharp[x := e'^\sharp]
\end{aligned}$$

as required.

Cases x **and** $e_1\ e_2$**:** Trivial.

Case $\lambda y\!:\!\tau.\,e$**:** our inductive hypothesis is that $\forall x, e'$, $(e[x := e'])^\sharp = e^\sharp[x := e'^\sharp]$. We have:

$$\begin{aligned}
((\lambda y\!:\!\tau.\,e)[x := e'])^\sharp &= (\lambda y\!:\!\tau.\,e[x := e'])^\sharp \\
&= \left(\lambda z\!:\!0_\tau.\,\lambda y\!:\!\mathfrak{b}(\tau_1).\,(e[x := e'])^\sharp\right)\ \tau^\sharp \\
&= \left(\lambda z\!:\!0_\tau.\,\lambda y\!:\!\mathfrak{b}(\tau_1).\,e^\sharp[x := e'^\sharp]\right)\ \tau^\sharp \qquad \text{by the inductive hypothesis} \\
&= \left(\left(\lambda z\!:\!0_\tau.\,\lambda y\!:\!\mathfrak{b}(\tau_1).\,e^\sharp\right)\ \tau^\sharp\right)[x := e'^\sharp] \\
&= (\lambda y\!:\!\tau.\,e)^\sharp[x := e'^\sharp]
\end{aligned}$$

as required.

□

Now, we prove that well-typed LF terms are translated to well-typed STLC terms, modulo the changes to the context required for the translation to make sense. We define those changes as follows.

Definition 10 ($\mathfrak{b}^+(\Gamma)$)**.** $\forall$ LF contexts Γ, define the STLC context $\mathfrak{b}^+(\Gamma)$ as follows:

$$\mathfrak{b}^+(\Gamma) \triangleq \{(x\!:\!\mathfrak{b}(\tau)) \mid (x\!:\!\tau) \in \Gamma\} \cup \{(x_X\!:\!\mathfrak{b}(\kappa)) \mid (X\!:\!\kappa) \in \Gamma\} \cup \{(y_\tau\!:\!0_\tau \to (\tau \to 0_\tau) \to 0_\tau) \mid \text{base types } \tau\}$$

where x_X and y_τ are the fresh term variables defined in Figure 7.

This lemma ensures that our translation produces well-typed STLC terms, so that we can guarantee that they are strongly normalizing.

Lemma 20 (Soundness of translation)**.** *The following two properties hold:*

1. *If* $\Gamma \vdash \tau\!:\!\kappa$*, then* $\mathfrak{b}^+(\Gamma) \vdash \tau^\sharp : \mathfrak{b}(\kappa)$*.*
2. *If* $\Gamma \vdash e\!:\!\tau$*, then* $\mathfrak{b}^+(\Gamma) \vdash e^\sharp : \mathfrak{b}(\tau)$*.*

Proof. The proof is by simultaneous induction on $\Gamma \vdash \tau\!:\!\kappa$ and $\Gamma \vdash e\!:\!\tau$, proving both properties.

Case KVar: We have $X\!:\!\kappa \in \Gamma$, so by Definition 10, we have $x_X\!:\!\mathfrak{b}(\kappa) \in \mathfrak{b}^+(\Gamma)$.
Since $X^\sharp = x_X$, we have $X^\sharp\!:\!\mathfrak{b}(\kappa) \in \mathfrak{b}^+(\Gamma)$ and we are done.

Case KPi: We have $\Gamma \vdash \tau_1 : *$ and $\Gamma, x : \tau_1 \vdash \tau_2 : *$. Our inductive hypotheses are that $\mathfrak{b}^+(\Gamma) \vdash \tau_1^\sharp : \mathfrak{b}(*)$ and $\mathfrak{b}^+(\Gamma, x\!:\!\tau_1) \vdash \tau_2^\sharp : \mathfrak{b}(*)$. We must show that $\mathfrak{b}^+(\Gamma) \vdash (\Pi x\!:\!\tau_1.\tau_2)^\sharp : \mathfrak{b}(*)$.
Note that $\mathfrak{b}(*) = 0_\tau$. Now, $(\Pi x\!:\!\tau_1.\tau_2)^\sharp = y_{\mathfrak{b}(\tau_1)}\ \tau_1^\sharp\ \left(\lambda x\!:\!\mathfrak{b}(\tau_1).\, \tau_2^\sharp\right)$.
By Definition 10, we have $y_{\mathfrak{b}(\tau_1)} : 0_\tau \to (\mathfrak{b}(\tau_1) \to 0_\tau) \to 0_\tau \in \mathfrak{b}^+(\Gamma)$.
By the first inductive hypothesis, $\mathfrak{b}^+(\Gamma) \vdash \tau_1^\sharp : 0_\tau$. Using the second inductive hypothesis, we see that $\mathfrak{b}^+(\Gamma) \vdash \lambda x\!:\!\mathfrak{b}(\tau_1).\, \tau_2^\sharp : \mathfrak{b}(\tau_1) \to 0_\tau$.
So the term $y_{\mathfrak{b}(\tau_1)}\ \tau_1^\sharp\ \left(\lambda x\!:\!\mathfrak{b}(\tau_1).\, \tau_2^\sharp\right)$ is a well-typed STLC application of overall type $0_\tau = \mathfrak{b}(*)$, as required.

Case KApp: We have $\Gamma \vdash \tau : \Pi x\!:\!\tau_1.\tau_2$ and $\Gamma \vdash e : \tau_1$. Our inductive hypotheses are that
$\mathfrak{b}^+(\Gamma) \vdash \tau^\sharp : \mathfrak{b}(\Pi x\!:\!\tau_1.\tau_2)$ and $\mathfrak{b}^+(\Gamma) \vdash e^\sharp : \mathfrak{b}(\tau_1)$. We must show that $\mathfrak{b}^+(\Gamma) \vdash (\tau\ e)^\sharp : \mathfrak{b}(\tau_2[x := e])$.
By Lemma 15, this is equivalent to showing that $\mathfrak{b}^+(\Gamma) \vdash (\tau\ e)^\sharp : \mathfrak{b}(\tau_2)$. We have $(\tau\ e)^\sharp = \tau^\sharp\ e^\sharp$, so we can apply the inductive hypotheses.
$\mathfrak{b}(\Pi x\!:\!\tau_1.\tau_2) = \mathfrak{b}(\tau_1) \to \mathfrak{b}(\tau_2)$, so by the first inductive hypothesis $\mathfrak{b}^+(\Gamma) \vdash \tau^\sharp : \mathfrak{b}(\tau_1) \to \mathfrak{b}(\tau_2)$.
Since $\mathfrak{b}^+(\Gamma) \vdash e^\sharp : \mathfrak{b}(\tau_1)$ by the second inductive hypothesis, we have a well-typed STLC application of overall type $\mathfrak{b}(\tau_2)$, as required.

Case KConstr: We have $\Gamma \vdash \tau_1 : *$ and $\Gamma, x : \tau_1 \vdash \tau_2 : \kappa$. The inductive hypotheses are that $\mathfrak{b}^+(\Gamma) \vdash \tau_1^\sharp : \mathfrak{b}(*)$ and $\mathfrak{b}^+(\Gamma, x\!:\!\tau_1) \vdash \tau_2^\sharp : \mathfrak{b}(\kappa)$. We must show that $\mathfrak{b}^+(\Gamma) \vdash (\lambda x\!:\!\tau_1.\, \tau_2)^\sharp : \mathfrak{b}(\Pi x\!:\!\tau_1.\kappa)$.
Now, $(\lambda x : \tau_1.\, \tau_2)^\sharp = \left(\lambda z\!:\!0_\tau.\, \lambda x\!:\!\mathfrak{b}(\tau_1).\, \tau_2^\sharp\right)\ \tau_1^\sharp$. Note that $\mathfrak{b}(*) = 0_\tau$. Thus, by the first inductive hypothesis, $\mathfrak{b}^+(\Gamma) \vdash \tau_1^\sharp : 0_\tau$, so we have a well-typed STLC application.
The overall STLC type is the type of $\lambda x\!:\!\mathfrak{b}(\tau_1).\, \tau_2^\sharp$, which by the second inductive hypothesis is $\mathfrak{b}(\tau_1) \to \mathfrak{b}(\kappa)$, as required.

Case KConv: This case follows directly by Lemma 18.

Case TVar: We have $x\!:\!\tau \in \Gamma$ and $\Gamma \vdash \tau : *$. By Definition 10, we have $x\!:\!\mathfrak{b}(\tau) \in \mathfrak{b}^+(\Gamma)$. Since $x^\sharp = x$, we have $x^\sharp : \mathfrak{b}(\tau) \in \mathfrak{b}^+(\Gamma)$, so that $\mathfrak{b}^+(\Gamma) \vdash x^\sharp : \mathfrak{b}(\tau)$ as required.

Case TAbs: We have $\Gamma \vdash \tau_1 : *$ and $\Gamma, x : \tau_1 \vdash e : \tau_2$. The inductive hypotheses are that $\mathfrak{b}^+(\Gamma) \vdash \tau_1^\sharp : \mathfrak{b}(*)$ and $\mathfrak{b}^+(\Gamma, x\!:\!\tau_1) \vdash e^\sharp : \mathfrak{b}(\tau_2)$. We must show that $\mathfrak{b}^+(\Gamma) \vdash (\lambda x\!:\!\tau_1.\, e)^\sharp : \mathfrak{b}(\Pi x\!:\!\tau_1.\tau_2)$.
Note that $\mathfrak{b}(*) = 0_\tau$ and $\mathfrak{b}(\Pi x\!:\!\tau_1.\tau_2) = \mathfrak{b}(\tau_1) \to \mathfrak{b}(\tau_2)$. We have $(\lambda x\!:\!\tau_1.\, e)^\sharp = \left(\lambda z\!:\!0_\tau.\, \lambda x\!:\!\mathfrak{b}(\tau_1).\, e^\sharp\right)\ \tau^\sharp$.
By the first inductive hypothesis, we have $\mathfrak{b}^+(\Gamma) \vdash \tau_1^\sharp : 0_\tau$, so this is a well-typed application.
Note also that by the second inductive hypothesis, $\mathfrak{b}^+(\Gamma, x\!:\!\tau_1) \vdash e^\sharp : \mathfrak{b}(\tau_2)$, so $\mathfrak{b}^+(\Gamma) \vdash \lambda x\!:\!\mathfrak{b}(\tau_1).\, e^\sharp : \mathfrak{b}(\tau_1) \to \mathfrak{b}(\tau_2)$.
So the overall type of the application $\left(\lambda z\!:\!0_\tau.\, \lambda x\!:\!\mathfrak{b}(\tau_1).\, e^\sharp\right)\ \tau^\sharp$ is $\mathfrak{b}(\tau_1) \to \mathfrak{b}(\tau_2)$, as required.

Case TApp: We have $\Gamma \vdash e_1 : \Pi x\!:\!\tau_1.\tau_2$ and $\Gamma \vdash e_2 : \tau_1$. The inductive hypotheses are that
$\mathfrak{b}^+(\Gamma) \vdash e_1^\sharp : \mathfrak{b}(\Pi x\!:\!\tau_1.\tau_2)$ and $\mathfrak{b}^+(\Gamma) \vdash e_2^\sharp : \mathfrak{b}(\tau_1)$. We must show that $\mathfrak{b}^+(\Gamma) \vdash (e_1\ e_2)^\sharp : \mathfrak{b}(\tau_2[x := e_2])$.
By Lemma 15, this is equivalent to showing that $\mathfrak{b}^+(\Gamma) \vdash (e_1\ e_2)^\sharp : \mathfrak{b}(\tau_2)$. We have $(e_1\ e_2)^\sharp = e_1^\sharp\ e_2^\sharp$, so we can apply the inductive hypotheses. Note that $\mathfrak{b}(\Pi x : \tau_1.\tau_2) = \mathfrak{b}(\tau_1) \to \mathfrak{b}(\tau_2)$, so by the first inductive hypothesis, $\mathfrak{b}^+(\Gamma) \vdash e_1^\sharp : \mathfrak{b}(\tau_1) \to \mathfrak{b}(\tau_2)$.
Since (by the second inductive hypothesis) $\mathfrak{b}^+(\Gamma) \vdash e_2^\sharp : \mathfrak{b}(\tau_1)$, the application $e_1^\sharp\ e_2^\sharp$ is a well-typed STLC application: $\mathfrak{b}^+(\Gamma) \vdash (e_1\ e_2)^\sharp : \mathfrak{b}(\tau_2)$, as required.

Case TConv: This case follows directly by Lemma 17.

□

Finally, we prove that our translation does not erase any β-reductions.

Lemma 21 (Translation preserves reductions)**.** *The following two properties hold:*

1. *If* $e_1 \to_\beta e_2$ *then* $e_1^\sharp \to_\beta^+ e_2^\sharp$.
2. *If* $\tau_1 \to_\beta \tau_2$ *then* $\tau_1^\sharp \to_\beta^+ \tau_2^\sharp$.

Proof. Simultaneous induction on $e_1 \to_\beta e_2$ and $\tau_1 \to_\beta \tau_2$. We omit the trivial cases which just lift a reduction in a subterm, considering only the two cases which perform a substitution.

Case $\tau\beta$**:** we have $(\lambda x\!:\!\tau_1.\, \tau_2)\ v \to_\beta \tau_2[x := v]$, where v is a term-level value.
We must show that $((\lambda x\!:\!\tau_1.\, \tau_2)\ v)^\sharp \to_\beta^+ (\tau_2[x := v])^\sharp$. We have:

$$((\lambda x\!:\!\tau_1.\, \tau_2)\ v)^\sharp = (\lambda x\!:\!\tau_1.\, \tau_2)^\sharp\ v^\sharp = \left(\left(\lambda z\!:\!0_\tau.\, \lambda x\!:\!\mathfrak{b}(\tau_1).\, \tau_2^\sharp\right)\ \tau_1^\sharp\right)\ v^\sharp \to_\beta \left(\lambda x\!:\!\mathfrak{b}(\tau_1).\, \tau_2^\sharp\right)\ v^\sharp \to_\beta \tau_2^\sharp[x := v^\sharp]$$

which by Lemma 19 is equivalent to $(\tau_2[x := v])^\sharp$. Note that we took two β-reductions in STLC where LF used only one. So our translation does not decrease the number of β-reductions in evaluating the term.

Case β**:** we have $(\lambda x\!:\!\tau.\, e)\ v \to_\beta e[x := v]$. We must show that $((\lambda x\!:\!\tau.\, e)\ v)^\sharp \to_\beta^+ (e[x := v])^\sharp$. We have:

$$((\lambda x\!:\!\tau.\, e)\ v)^\sharp = (\lambda x\!:\!\tau.\, e)^\sharp\ v^\sharp = \left(\left(\lambda z\!:\!0_\tau.\, \lambda x\!:\!\mathfrak{b}(\tau).\, e^\sharp\right)\ \tau^\sharp\right)\ v^\sharp \to_\beta \left(\lambda x\!:\!\mathfrak{b}(\tau).\, e^\sharp\right)\ v^\sharp \to_\beta e^\sharp[x := v^\sharp]$$

which by Lemma 19 is equivalent to $(e[x := v])^\sharp$. Again, note that we took two β-reductions in STLC where LF used only one. So our translation does not decrease the number of β-reductions in evaluating the term.

□

Theorem 4 (Strong normalization)**.** *Every well-typed term in* λP *is strongly normalizing.*

Proof. By contradiction.

Let e be a λP term s.t. $\vdash e : \tau$ for some well-kinded type τ. Suppose (for contradiction) that e is not strongly normalizing. By type safety of λP, we know that evaluation of e never gets stuck. So the only possibility is that there exists some infinite sequence of LF β-reductions from e. Let these β-reductions be $e \to_\beta e_1 \to_\beta e_2 \to_\beta \cdots$. Now, applying Lemma 21 to each of the β-reductions, we have that $e^\sharp \to_\beta^+ e_1^\sharp \to_\beta^+ e_2^\sharp \to_\beta^+ \cdots$, i.e. there exists an infinite sequence of STLC β-reductions starting from $e^\sharp$. But this is impossible since the STLC is strongly normalizing (Theorem 1). We have our contradiction.

Therefore, every well-typed term in λP is strongly normalizing. □

An alternative proof of strong normalization for LF, which uses the same overall technique but a subtly different translation, may be found in the Appendix of Harper [2].

6 Calculus of Constructions: Higher Order Dependent Type System

The Calculus of Constructions is the "highest" calculus in the lambda cube, combining type constructors (types depend on types), polymorphism (terms depend on types), and dependent types (types depend on terms). With the addition of inductive data types, it forms the calculus underlying the Coq proof assistant. We eschew a formal definition of the calculus of constructions here.

We can prove that the calculus of constructions is strongly normalizing. Casighino [3] gives three proofs of strong normalization: a proof by reduction-preserving translation to System F$_\omega$, a proof by an extension of the method of candidates, and a proof by means of a "realizability semantics". We consider only the first two proofs.

The first proof of Casighino [3] is by reduction-preserving translation, with System F$_\omega$ as the target calculus; this proof thus relies on Theorem 3. We begin with a contracting map $\mathfrak{b}$, which maps CoC types to F$_\omega$ types by erasing type dependency, thus losing type-level reductions. We also give a translation $e^\sharp$, which translates CoC types and terms to F_ω terms by lowering type-level abstractions to the level of terms. (Casighino uses the syntax $[\![\cdot]\!]$ for $\flat$ and $[\cdot]$ for $e^\sharp$.) The overall soundness theorem is analogous to Lemma 20, stating that if $\Gamma \vdash e : \tau$ in CoC, then $\flat(\Gamma) \vdash e^\sharp : \flat(\tau)$ in F_ω. An additional complication arises from the fact that in CoC, term and type variables are less clearly separated than in F_ω. The solution to this is to index $\flat$ and $e^\sharp$ by contexts, which keep track of whether variables are at the term or type level. We omit the full details of the translation, but it is similar to that of Figure 7. We again finish the proof by showing a lemma analogous to Lemma 21. A similar proof is given in Sørensen [1].

The second proof, based on that of [4], is similar to the method of candidates proof. We introduce the concept of a *key redex*, which is the "first" redex which must be reduced in an term: if a term e is reduced without reducing its key redex, the key redex remains in place. With this definition of key redex, we modify the third condition of Definition 2 to instead state that if term e is strongly normalizing and the reduction of the key redex of e is in C, then the key redex of e is in C. The reason to define key redexes as opposed to the simpler characterization of Definition 2 is that key redexes can occur at any level, not just term-to-term. Having defined candidates, we again define our sets $[\![\cdot]\!]_\xi$ to map types to candidates, but we must also define mappings from kinds to candidates which parameterize the $[\![\cdot]\!]_\xi$ sets. Having done this, we prove a number of helper lemmas, including one analogous to Lemma 7. These allow us to complete the strong normalization proof.

In both cases, the proofs of strong normalization require a number of "administrative" modifications to the definitions in Sections 5 and 3 respectively, to keep track of the more complex ways in which terms can reduce in the CoC. However, the essential aspects of both proof techniques are preserved. This illustrates that both the reduction-preserving translation technique and the method of candidates are suitable for proving strong normalization of calculi with various forms of dependency.

7 Conclusions and future work

In this report, we investigated proof techniques for proving strong normalization of complex calculi from the lambda cube, including dependently typed calculi. We gave explicit proofs of strong normalization for System F and LF, and sketched proof techniques for System F_ω and the calculus of constructions. In particular, we investigated two proof techniques: the method of candidates, which generalizes logical relations to account for complex dependencies among types and terms; and reduction-preserving translation, which allows us to lift a strong normalization proof from a simple calculus to a more complex one.

We find several areas for further investigation. In particular, we expect it to be possible to prove strong normalization for LF by the method of candidates, given that such a proof exists for the more complex calculus of constructions. It will be interesting to understand what definitions of candidates are suitable for that proof. More generally, we are interested in how these proof techniques apply to proofs of strong normalization for the other calculi in the lambda cube. Although those calculi are less prevalent in practice, it will be interesting to investigate which axes of the lambda cube produce difficulties for the two proof techniques. Furthermore, we would like to understand whether, and how, these techniques can generalize from strong normalization to prove other properties of polymorphic or depenently-typed lambda calculi.

References

[1] M.H. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism.* Studies in Logic and the Foundations of Mathematics. Elsevier, 2006.

[2] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, January 1993.

[3] Chris Casinghino. Strong normalization for the calculus of constructions, 2010. Technical Report.

[4] J.H. Geuvers. *A short and flexible proof of strong normalization for the calculus of constructions.* Computing science reports. Technische Universiteit Eindhoven, 1994.