Using the Hereditary Substitution Function in Normalization Proofs

Harley Eades and Aaron Stump

Computer Science
The University of Iowa

Qualifying Exam 2011

Introduction

- What is a functional programming language?
- The λ-calculus.
  - The language, operational semantics, and examples.
  - Paradoxes and the need for something better.
- The Simply Typed λ-calculus.
  - Language, types, and examples.
- A bit about logic.
  - Intuitionistic logic and how type theories can be considered intuitionistic logics.
  - The normalization property.
- The hereditary substitution function.
  - The definition and properties of the function.
- Normalization by hereditary substitution.
  - Semantics, a main substitution lemma, and type soundness.
- Normalization of STLC, SSF, and SSF$^{\omega}$.
  - Define each language and apply normalization by hereditary substitution.

What is a functional programming language?

- A functional programming language is a programming language that is based on a mathematical foundation.
- This foundation is the λ-calculus.
- A few of the most popular functional programming languages are ML, Haskell, and (pure) Scheme.

The λ-Calculus

Definition (The Syntax)

The language of the λ-calculus consists of only variables, functions, and applications. The grammar is as follows:

$$t \quad ::= \quad x \quad | \quad \lambda x.t \quad | \quad t\ t$$

Definition (The Operational Semantics)

The operational semantics for the λ-calculus is the following:

$$(\lambda x.t)\ t' \rightsquigarrow [t'/x]t$$

The λ-Calculus

Definition

The capture avoiding substitution function is defined by induction on the form of t', the term we are substituting into.

$$[t/x]x = t$$

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

$$[t/x](\lambda x.t') = \lambda x.t'$$

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

Where $y \notin FV(t)$.

$$[t/x](\lambda y.t') = \lambda y.[([z/y]t)/x]t'$$

Where $y \in FV(t)$ and z is a variable distinct from all variables (free or bound) in t.

$$[t/x](t_1\ t_2) = ([t/x]t_1)\ ([t/x]t_2)$$

The λ-Calculus

- Example terms:

Identity Function:	$\lambda x.x$
Squaring Function:	$\lambda x.x\ x$
True:	$\lambda x.\lambda y.x$
False:	$\lambda x.\lambda y.y$
Conjunction:	$\lambda x.\lambda y.x\ y\ x$
Disjunction:	$\lambda x.\lambda y.x\ x\ y$
Zero:	$\lambda s.\lambda z.z$
One:	$\lambda s.\lambda z.s\ z$
Plus:	$\lambda n_1.\lambda n_2.\lambda s.\lambda z.n_1\ s\ (n_2\ s\ z)$
Multiplication:	$\lambda n_1.\lambda n_2.\lambda s.\lambda z.n_2\ (plus\ n_1)\ z$

The λ-Calculus

- Example computation $(3 + 2)$:
  Let $3 = \lambda s.\lambda z.s\ (s\ (s\ z))$, $2 = \lambda s.\lambda z.s\ (s\ z)$, and
  $plus = \lambda n_1.\lambda n_2.\lambda s.\lambda z.n_1\ s\ (n_2\ s\ z)$. Then

$$
\begin{array}{lll}
(plus\ 3)\ 2 & \equiv & ((\lambda n_1.\lambda n_2.\lambda s.\lambda z.n_1\ s\ (n_2\ s\ z))\ 3)\ 2 \\
& \rightsquigarrow_\beta & (\lambda n_2.\lambda s.\lambda z.3\ s\ (n_2\ s\ z))\ 2 \\
& \rightsquigarrow_\beta & \lambda s.\lambda z.3\ s\ (2\ s\ z) \\
& \equiv & \lambda s.\lambda z.(\lambda s.\lambda z.s\ (s\ (s\ z)))\ s\ (2\ s\ z) \\
& \rightsquigarrow_\beta & \lambda s.\lambda z.(\lambda z.s\ (s\ (s\ z)))\ (2\ s\ z) \\
& \equiv & \lambda s.\lambda z.(\lambda z.s\ (s\ (s\ z)))\ ((\lambda s.\lambda z.s\ (s\ z))\ s\ z) \\
& \rightsquigarrow_\beta & \lambda s.\lambda z.(\lambda z.s\ (s\ (s\ z)))\ ((\lambda z.s\ (s\ z))\ z) \\
& \rightsquigarrow_\beta & \lambda s.\lambda z.(\lambda z.s\ (s\ (s\ z)))\ (s\ (s\ z)) \\
& \rightsquigarrow_\beta & \lambda s.\lambda z.s\ (s\ (s\ (s\ (s\ z)))) \\
& \equiv & 5.
\end{array}
$$

Paradoxes of the λ-Calculus

- Infinite loop:

$$(\lambda x.x\ x)\ (\lambda x.x\ x) \rightsquigarrow_\beta (\lambda x.x\ x)\ (\lambda x.x\ x) \rightsquigarrow_\beta \cdots .$$

- Loops are good for general purpose programming, but not for logic.
  - Terms like the one above allows the formulation of paradoxes in the λ-calculus.
  - Thus, the λ-calculus is inconsistent as a logic.
  - Church fixed this by adding types.

The Simply Typed λ-calculus

Definition (The Syntax)

The grammar is as follows:

$$\begin{array}{lllllll} t & ::= & x & \mid & \lambda x : \phi.t & \mid & t\ t \\ \phi & ::= & X & \mid & \phi \to \phi & & \\ \Gamma & ::= & \cdot & \mid & \Gamma, x : \phi & & \end{array}$$

Definition (Type Checking Rules)

$$\frac{\Gamma(x) = \phi}{\Gamma \vdash x : \phi}\ \textsc{Var} \qquad \frac{\Gamma, x : \phi_1 \vdash t : \phi_2}{\Gamma \vdash \lambda x : \phi_1.t : \phi_1 \to \phi_2}\ \textsc{Lam} \qquad \frac{\Gamma \vdash t_1 : \phi_1 \to \phi_2 \qquad \Gamma \vdash t_2 : \phi_1}{\Gamma \vdash t_1\ t_2 : \phi_2}\ \textsc{App}$$

Example Typing Derivations

$$\dfrac{\dfrac{\dfrac{\dfrac{}{s : X \to X, z : X \vdash s : X \to X}\ \textsc{Var} \qquad \dfrac{}{s : X \to X, z : X \vdash z : X}\ \textsc{Var}}{s : X \to X, z : X \vdash s\ z : X}\ \textsc{App}}{s : X \to X \vdash \lambda z : X.s\ z : X \to X}\ \textsc{Lam}}{\cdot \vdash \lambda s : X \to X.\lambda z : X.s\ z : (X \to X) \to X \to X}\ \textsc{Lam}$$

$$\dfrac{\dfrac{\dfrac{}{x : X \to X \vdash x : X \to X}\ \textsc{Var} \qquad \dfrac{}{x : X \to X \vdash x : X}\ ???}{x : X \to X \vdash x\ x : X \to X}\ \textsc{App}}{\cdot \vdash \lambda x : X \to X.x\ x : X \to X}\ \textsc{Lam}$$

A Bit about Logic

- Type theories like STLC can be considered as intuitionistic logics.
- In fact there is a one-to-one correspondence between STLC and minimal intuitionistic propositional logic.
- This correspondence is called the Curry-Howard correspondence or proofs-as-programs and propositions-as-types correspondence.
- We reveal this correspondence by showing how STLC and minimal intuitionistic propositional logic correspond using an interpretation called the BHK-interpretation.

Minimal Intuitionistic Propositional Logic

Definition (Gentzen's Natural Deduction)

We denote propositional variables by x, x_i, y, and so on. We assume an infinite number of them. All formulas will be denoted by ϕ_i. We denote sets of assumptions by Γ_i.

$$\phi \quad ::= \quad x \quad | \quad \phi_1 \to \phi_2$$

$$\frac{}{\phi \vdash \phi}\ \mathrm{I} \qquad \frac{\Gamma, \phi_1 \vdash \phi_2}{\Gamma \vdash \phi_1 \to \phi_2} \to_i \qquad \frac{\Gamma_1 \vdash \phi_1 \to \phi_2 \qquad \Gamma_2 \vdash \phi_1}{\Gamma_1, \Gamma_2 \vdash \phi_2} \to_e$$

The BHK-Interpretation

- In intuitionistic logic or constructive logic proofs of propositions must be constructed.
- The Brouwer, Heyting, and Kolmogorov interpretation (BHK-interpretation) tells us exactly how to construct the proof of a proposition in minimal intuitionistic logic.

Definition

The BHK-interpretation:

$c\, r\, (\phi_1 \to \phi_2)$ iff c is a function, $\lambda x.t$, such that for any $d\, r\, \phi_1$ $(\lambda x.t)\, d\, r\, \phi_2$.

We say a construction c realizes ϕ iff $c\, r\, \phi$.

Curry-Howard Correspondence

- Through the work of Curry, Howard, Tait, Laüchli, De Bruijn, and Prawitz there exists an important correspondence between type theory and intuitionistic logic stated as follows:

  Propositions = Types and Proofs = Programs.

- Consider the type-checking rules for STLC:

$$\frac{\Gamma(x) = \phi}{\Gamma \vdash x : \phi}\ \textsc{Var} \qquad \frac{\Gamma, x : \phi_1 \vdash t : \phi_2}{\Gamma \vdash \lambda x : \phi_1.t : \phi_1 \to \phi_2}\ \textsc{Lam} \qquad \frac{\Gamma \vdash t_1 : \phi_1 \to \phi_2 \qquad \Gamma \vdash t_2 : \phi_1}{\Gamma \vdash t_1\ t_2 : \phi_2}\ \textsc{App}$$

Curry-Howard Correspondence

- Through the work of Curry, Howard, Tait, Laüchli, De Bruijn, and Prawitz there exists an important correspondence between type theory and intuitionistic logic stated as follows:

  Propositions = Types and Proofs = Programs.

- Consider the type-checking rules for STLC:

$$\frac{\Gamma(x) = \phi}{\Gamma \vdash x : \phi}\ \textsc{Var} \qquad \frac{\Gamma, x : \phi_1 \vdash t : \phi_2}{\Gamma \vdash \lambda x : \phi_1 . t : \phi_1 \to \phi_2}\ \textsc{Lam} \qquad \frac{\Gamma \vdash t_1 : \phi_1 \to \phi_2 \qquad \Gamma \vdash t_2 : \phi_1}{\Gamma \vdash t_1\ t_2 : \phi_2}\ \textsc{App}$$

- Consider the type-checking rules for STLC:

$$\frac{}{\Gamma', \phi \vdash \phi}\ \textsc{Var} \qquad \frac{\Gamma', \phi_1 \vdash \phi_2}{\Gamma' \vdash \phi_1 \to \phi_2}\ \textsc{Lam} \qquad \frac{\Gamma' \vdash \phi_1 \to \phi_2 \qquad \Gamma' \vdash \phi_1}{\Gamma' \vdash \phi_2}\ \textsc{App}$$

Proofs as Programs

$$
\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\begin{array}{l}(p \to q), (q \to u), p \models p \\ (p \to q), (q \to u), p \models p \to q\end{array}}{(p \to q), (q \to u), p \models q}\to_e \quad (p \to q), (q \to u), p \models q \to u}{(p \to q), (q \to u), p \models u}\to_e}{(p \to q), (q \to u) \models p \to u}\to_i}{(p \to q) \models (q \to u) \to (p \to u)}\to_i}{\cdot \models (p \to q) \to (q \to u) \to (p \to u)}\to_i
$$

$$
\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\begin{array}{l}p : (P \to Q), q : (Q \to U), z; P \vdash z : P \\ p : (P \to Q), q : (Q \to U), z; P \vdash p : (P \to Q)\end{array}}{p : (P \to Q), q : (Q \to U), z; P \vdash p\, z : Q} \quad p : (P \to Q), q : (Q \to U), z; P \vdash q : Q \to U}{p : (P \to Q), q : (Q \to U), z : P \vdash q\,(p\, z) : U}}{p : (P \to Q), q : (Q \to U) \vdash \lambda z : P.q\,(p\, z) : (P \to U)}}{p : (P \to Q) \vdash \lambda q : (Q \to U).\lambda z : P.q\,(p\, z) : (Q \to U) \to (P \to U)}}{\cdot \vdash \lambda p : (P \to Q).\lambda q : (Q \to U).\lambda z : P.q\,(p\, z) : (P \to Q) \to (Q \to U) \to (P \to U)}
$$

The Normalization Property

- The property where there exists a computation path (w.r.t the operational semantics) where all programs definable in a typed λ-calculi terminate. More precisely, $\forall t.\exists t'.t \rightsquigarrow^* t' \not\rightsquigarrow$. We call t' a normal form.
- The normalization property is important, because proofs of logical formulas must be finite and total.
- Diverging proofs do not establish any kind of truth.
- Normalization is not a trivial property and is often very difficult to prove.
  - The property is a meta-level property which requires a strong meta-theory.
  - The complexity of normalization proofs is the driving force behind this research.
    - Existing proof methods are hard to use, even for weak theories.

Hereditary substitution function [Watkins et al., 2004]

- Watkins et al. defined a dependently typed programming language called Canonical LF (CLF).
  - The language only consisted of normal forms.
  - This prevented capture avoiding substitution from being used by the operational semantics.
    - Example: $(\lambda x : X \to X.x\ y)(\lambda x : X.x) \leadsto [(\lambda x : X.x)/x](x\ y)$.
- Syntax: $[t/x]^{\phi} t' = t''$.
- Like ordinary capture avoiding substitution.
- Except, if the substitution introduces a redex, then that redex is recursively reduced.
  - Example: $[(\lambda z : X.z)/x]^{X \to X}(x\ y)\ (\leadsto (\lambda z : X.z)y \leadsto [y/z]^{X} z) = y$.

Normalization by hereditary substitution

- Proving normalization of some type theory using hereditary substitution involves six main steps:
  i. define a well-founded ordering on types,
  ii. define the hereditary substitution function,
  iii. prove the properties of the hereditary substitution function,
  iv. define a semantics for types called the interpretation of types,
  v. prove the semantics is closed under hereditary substitutions (this implies that the semantics is closed under capture avoiding substitutions), and
  vi. prove all typeable terms are members of the interpretation of their type. This is known as type soundness.

Normalization of STLC

- The ordering on types is just the strict subexpression ordering.
  - I.e. $\phi_1 \to \phi_2 >_\Gamma \phi_i$ where $i \in \{1, 2\}$.

Definition (The Construct Type Function)

$$ctype_\phi(x, x) = \phi$$

$$ctype_\phi(x, t_1\ t_2) = \phi''$$

Where $ctype_\phi(x, t_1) = \phi' \to \phi''$.

Lemma (Properties of $ctype_\phi$)

i. *If $ctype_\phi(x, t) = \phi'$ then $head(t) = x$ and ϕ' is a subexpression of ϕ.*
ii. *If $\Gamma, x : \phi, \Gamma' \vdash t : \phi'$ and $ctype_\phi(x, t) = \phi''$ then $\phi' \equiv \phi''$.*

Normalization of STLC

Definition (The Hereditary Substitution Function for STLC)

$[t/x]^{\phi} x = t$

$[t/x]^{\phi} y = y$

Where y is a variable distinct from x.

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

$[t/x]^{\phi}(t_1\ t_2) = ([t/x]^{\phi} t_1)\ ([t/x]^{\phi} t_2)$

Where $([t/x]^{\phi} t_1)$ is not a λ-abstraction, or both $([t/x]^{\phi} t_1)$ and t_1 are λ-abstractions, or $ctype_{\phi}(x, t_1)$ is undefined.

$[t/x]^{\phi}(t_1\ t_2) = [([t/x]^{\phi} t_2)/y]^{\phi''} s'_1$

Where $([t/x]^{\phi} t_1) \equiv \lambda y : \phi''.s'_1$ for some y, s'_1, and ϕ'' and $ctype_{\phi}(x, t_1) = \phi'' \to \phi'$.

Lemma (Properties of $ctype_{\phi}$)

iii. *If $\Gamma, x : \phi, \Gamma' \vdash t_1\ t_2 : \phi'$, $\Gamma \vdash t : \phi$, $[t/x]^{\phi} t_1 = \lambda y : \phi_1.q$, and t_1 is not then there exists a type ψ such that $ctype_{\phi}(x, t_1) = \psi$.*

Normalization of STLC

Lemma (Total and Type Preserving)

Suppose $\Gamma \vdash t : \phi$ *and* $\Gamma, x : \phi, \Gamma' \vdash t' : \phi'$. *Then there exists a term* t'' *such that* $[t/x]^{\phi} t' = t''$ *and* $\Gamma, \Gamma' \vdash t'' : \phi'$.

Lemma (Normality Preserving)

If $\Gamma \vdash n : \phi$ *and* $\Gamma, x : \phi \vdash n' : \phi'$ *then there exists a normal term* n'' *such that* $[n/x]^{\phi} n' = n''$.

Lemma (Soundness with Respect to Reduction)

If $\Gamma \vdash t : \phi$ *and* $\Gamma, x : \phi, \Gamma' \vdash t' : \phi'$ *then* $[t/x]t' \rightsquigarrow^{*} [t/x]^{\phi} t'$.

Normalization of STLC

Lemma (Redex Preserving)

If $\Gamma \vdash t : \phi$*,* $\Gamma, x : \phi, \Gamma' \vdash t' : \phi'$ *then* $|rset(t', t)| \geq |rset([t/x]^{\phi} t')|$*.*

- We call this property redex preservation, because eventually we would like to characterize which redexes are actually destroyed and which remain. In particularly the latter.

Normalization of STLC

Definition

First we define when a normal form is a member of the interpretation of type ϕ in context Γ

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

and this definition is extended to non-normal forms in the following way

$$t \in [\![\phi]\!]_\Gamma \iff t \rightsquigarrow^! n \in [\![\phi]\!]_\Gamma,$$

where $t \rightsquigarrow^! t'$ is syntactic sugar for $t \rightsquigarrow^* t' \not\rightsquigarrow$.

Lemma (Substitution for the Interpretation of Types)

If $n' \in [\![\phi']\!]_{\Gamma,x:\phi,\Gamma'}$, $n \in [\![\phi]\!]_\Gamma$, then $[n/x]^\phi n' \in [\![\phi']\!]_{\Gamma,\Gamma'}$.

Normalization of STLC

Theorem (Type Soundness)

If $\Gamma \vdash t : \phi$ then $t \in [\![\phi]\!]_\Gamma$.

Corollary (Normalization)

If $\Gamma \vdash t : \phi$ then $t \rightsquigarrow^! n$.

Stratified System F

- Example: $\lambda X : *_l.\lambda x : X.x$.
- Example: $\forall X.\phi$.

Definition (Syntax for SSF)

$$\begin{array}{lll} K & := & *_0 \mid *_1 \mid \ldots \\ \phi & := & X \mid \phi \to \phi \mid \forall X : K.\phi \\ t & := & x \mid \lambda x : \phi.t \mid t\ t \mid \Lambda X : K.t \mid t[\phi] \end{array}$$

Definition (The Operational Semantics for SSF)

$$\begin{array}{lll} (\Lambda X : *_p.t)[\phi] & \rightsquigarrow & [\phi/X]t \\ (\lambda x : \phi.t)t' & \rightsquigarrow & [t'/x]t \end{array}$$

Stratified System F

Definition (Kinding Rules)

$$\frac{\Gamma \vdash \phi_1 : *_p \qquad \Gamma \vdash \phi_2 : *_q}{\Gamma \vdash \phi_1 \to \phi_2 : *_{max(p,q)}} \qquad \frac{\Gamma, X : *_q \vdash \phi : *_p}{\Gamma \vdash \forall X : *_q.\phi : *_{max(p,q)+1}} \qquad \frac{\Gamma(X) = *_p \qquad p \leq q \qquad \Gamma\ Ok}{\Gamma \vdash X : *_q}$$

Definition (Type-checking Rules for SSF)

$$\frac{\Gamma(x) = \phi \qquad \Gamma\ Ok}{\Gamma \vdash x : \phi} \qquad \frac{\Gamma, x : \phi_1 \vdash t : \phi_2}{\Gamma \vdash \lambda x : \phi_1.t : \phi_1 \to \phi_2} \qquad \frac{\Gamma \vdash t_1 : \phi_1 \to \phi_2 \quad \Gamma \vdash t_2 : \phi_1}{\Gamma \vdash t_1\ t_2 : \phi_2}$$

$$\frac{\Gamma, X : *_p \vdash t : \phi}{\Gamma \vdash \Lambda X : *_p.t : \forall X : *_p.\phi} \qquad \frac{\Gamma \vdash t : \forall X : *_l.\phi_1 \qquad \Gamma \vdash \phi_2 : *_l}{\Gamma \vdash t[\phi_2] : [\phi_2/X]\phi_1}$$

Normalization of SSF

Definition (Ordering on Types)

The ordering $>_\Gamma$ is defined as the least relation satisfying the universal closures of the following formulas:

$$\begin{array}{lcl} \phi_1 \to \phi_2 & >_\Gamma & \phi_1 \\ \phi_1 \to \phi_2 & >_\Gamma & \phi_2 \\ \forall X : *_l.\phi & >_\Gamma & [\phi'/X]\phi \text{ where } \Gamma \vdash \phi' : *_l. \end{array}$$

Lemma (Transitivity of $>_\Gamma$)

Let ϕ, ϕ', and ϕ'' be kindable types. If $\phi >_\Gamma \phi'$ and $\phi' >_\Gamma \phi''$ then $\phi >_\Gamma \phi''$.

Theorem (Well-founded ordering)

The ordering $>_\Gamma$ is well-founded on types ϕ such that $\Gamma \vdash \phi : *_l$ for some l.

Normalization of SSF

Definition

The construct type function for SSF is defined as follows:

$$ctype_{\phi}(x, x) = \phi$$

$$ctype_{\phi}(x, t_1\ t_2) = \phi''$$
Where $ctype_{\phi}(x, t_1) = \phi' \to \phi''$.

$$ctype_{\phi}(x, t[\phi']) = [\phi'/X]\phi''$$
Where $ctype_{\phi}(x, t) = \forall X : *_l.\phi''$.

Normalization of SSF

Definition (The Hereditary Substitution Function for SSF)

$[t/x]^{\phi} x = t$

$[t/x]^{\phi} y = y$
Where y is a variable distinct from x.

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

$[t/x]^{\phi}(\Lambda X : *_l.t') = \Lambda X : *_l.([t/x]^{\phi} t')$

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

$[t/x]^{\phi}(t_1\ t_2) = [([t/x]^{\phi} t_2)/y]^{\phi''} s'_1$
Where $([t/x]^{\phi} t_1) \equiv \lambda y : \phi''.s'_1$ for some y, s'_1, and ϕ'' and $ctype_{\phi}(x, t_1) = \phi'' \rightarrow \phi'$.

$[t/x]^{\phi}(t'[\phi']) = ([t/x]^{\phi} t')[\phi']$
Where $[t/x]^{\phi} t'$ is not a type abstraction or t' and $[t/x]^{\phi} t'$ are type abstractions.

$[t/x]^{\phi}(t'[\phi']) = [\phi'/X]s'_1$
Where $[t/x]^{\phi} t' \equiv \Lambda X : *_l.s'_1$, for some X, s'_1 and $\Gamma \vdash \phi' : *_q$, such that, $q \leq l$ and $ctype_{\phi}(x, t') = \forall X : *_l.\phi''$.

Normalization of SSF

Lemma (Properties of $ctype_\phi$)

i. If $\Gamma, x : \phi, \Gamma' \vdash t : \phi'$ and $ctype_\phi(x, t) = \phi''$ then $head(t) = x$, $\phi' \equiv \phi''$, and $\phi' \leq_{\Gamma,\Gamma'} \phi$.

ii. If $\Gamma, x : \phi, \Gamma' \vdash t_1\ t_2 : \phi'$, $\Gamma \vdash t : \phi$, $[t/x]^\phi t_1 = \lambda y : \phi_1.q$, and t_1 is not then there exists a type ψ such that $ctype_\phi(x, t_1) = \psi$.

iii. If $\Gamma, x : \phi, \Gamma' \vdash t'[\phi''] : \phi'$, $\Gamma \vdash t : \phi$, $[t/x]^\phi t' = \Lambda X : *_l.t''$, and t' is not then there exists a type ψ such that $ctype_\phi(x, t') = \psi$.

Normalization of SSF

- All the properties of the hereditary substitution function are exactly the same as for STLC. Their proofs only differ.
- The interpretation of types is exactly as for STLC.

Lemma (Substitution for the Interpretation of Types)

If $n' \in [\![\phi']\!]_{\Gamma,x:\phi,\Gamma'}$, $n \in [\![\phi]\!]_{\Gamma}$, *then* $[n/x]^{\phi} n' \in [\![\phi']\!]_{\Gamma,\Gamma'}$.

Theorem (Type Soundness)

If $\Gamma \vdash t : \phi$ *then* $t \in [\![\phi]\!]_{\Gamma}$.

Corollary (Normalization)

If $\Gamma \vdash t : \phi$ *then* $t \rightsquigarrow^{!} n$.

Stratified System F^ω

Definition (Syntax for SSF^ω)

$$
\begin{array}{lll}
K & := & K \to K \mid *_0 \mid *_1 \mid \ldots \\
\phi & := & X \mid \phi \to \phi \mid \forall X : K.\phi \mid \lambda X : *_l.\phi \mid \phi\ \phi \\
t & := & x \mid \lambda x : \phi.t \mid t\ t \mid \Lambda X : K.t \mid t[\phi]
\end{array}
$$

Definition (Operational Semantics for SSF^ω)

$$
\begin{array}{lll}
(\Lambda X : K.t)[\phi] & \rightsquigarrow & [\phi/X]t \\
(\lambda x : \phi.t)\ t' & \rightsquigarrow & [t'/x]t \\
(\lambda X : *_l.\phi)\ \phi' & \rightsquigarrow & [\phi'/x]\phi
\end{array}
$$

Stratified System F^{ω}

Definition (Kinding Rules)

$$\frac{\Gamma \vdash \phi_1 : *_p \qquad \Gamma \vdash \phi_2 : *_q}{\Gamma \vdash \phi_1 \to \phi_2 : *_{max(p,q)}}$$

$$\frac{\Gamma, X : *_q \vdash \phi : *_p}{\Gamma \vdash \forall X : *_q.\phi : *_{max(p,q)+1}}$$

$$\frac{\Gamma\ Ok \quad p \leq q \qquad \Gamma(X) = *_p}{\Gamma \vdash X : *_q}$$

$$\frac{\Gamma, X : K_1 \vdash \phi : K_2}{\Gamma \vdash \lambda X : K_1.\phi : K_1 \to K_2}$$

$$\frac{\Gamma \vdash \phi_1 : K_1 \to K_2 \qquad \Gamma \vdash \phi_2 : K_1}{\Gamma \vdash \phi_1\ \phi_2 : K_2}$$

Definition (Type-Checking Rules)

$$\frac{\Gamma(x) = \phi \qquad \Gamma\ Ok}{\Gamma \vdash x : \phi}$$

$$\frac{\Gamma, x : \phi_1 \vdash t : \phi_2}{\Gamma \vdash \lambda x : \phi_1.t : \phi_1 \to \phi_2}$$

$$\frac{\Gamma \vdash t_1 : \phi_1 \to \phi_2 \quad \Gamma \vdash t_2 : \phi_1}{\Gamma \vdash t_1\ t_2 : \phi_2}$$

$$\frac{\Gamma, X : *_p \vdash t : \phi}{\Gamma \vdash \Lambda X : *_p.t : \forall X : *_p.\phi}$$

$$\frac{\Gamma \vdash t : \forall X : *_l.\phi_1 \qquad \Gamma \vdash \phi_2 : *_l}{\Gamma \vdash t[\phi_2] : [\phi_2/X]\phi_1}$$

Normalization of SSF^{ω}

- To prove normalization of SSF^{ω} we must first prove normalization of the type level and then use this knowledge to prove normalization of the term (program) level.
- Normalization of the type level amounts to simply proving normalization of STLC.
- Normalization of the term level is essentially just normalization of SSF with a new type soundness result.

Normalization of SSF$^{\omega}$

Definition (The Construct Kind Function)

$$ckind_K(X, X) = K$$

$$ckind_K(X, \phi_1\ \phi_2) = K'$$
Where $ckind_K(X, \phi_1) = K'' \rightarrow K'$.

- The construct kind function has all the exact same properties as the construct type function for STLC.

Normalization of SSF$^\omega$

Definition (Type-Level Hereditary Substitution Function)

$$\{\phi/X\}^K X = \phi$$

$$\{\phi/X\}^K Y = Y$$

Where Y is a variable distinct from X.

$$\{\phi/X\}^K(\phi_1 \to \phi_2) = (\{\phi/X\}^K \phi_1) \to (\{\phi/X\}^K \phi_2)$$

$$\{\phi/X\}^K(\forall Y : *_l.\phi') = \forall Y : *_l.\{\phi/X\}^K \phi'$$

$$\{\phi/X\}^K(\lambda Y : K_1.\phi') = \lambda Y : K_1.(\{\phi/X\}^K \phi')$$

$$\{\phi/X\}^K(\phi_1\ \phi_2) = (\{\phi/X\}^K \phi_1)\ (\{\phi/X\}^K \phi_2)$$

Where $(\{\phi/X\}^K \phi_1)$ is not a λ-abstraction, or both $(\{\phi/X\}^K \phi_1)$ and ϕ_1 are λ-abstractions, or $ckind_K(X, \phi_1)$ is undefined.

$$\{\phi/X\}^K(\phi_1\ \phi_2) = \{(\{\phi/X\}^K \phi_2)/y\}^{K''} \phi'_1$$

Where $(\{\phi/X\}^K \phi_1) \equiv \lambda Y : K''.\phi'_1$ for some Y, ϕ'_1, and K'' and $ckind_K(X, \phi_1) = K'' \to K'$.

Normalization of SSF$^\omega$

- The type-level hereditary substitution function has all the exact same properties as the hereditary substitution function for STLC.
- Concluding normalization for the type-level is again identical to STLC.
- All that is left is concluding normalization of the term level.

Definition

First we define when a normal form is a member of the interpretation of normal type ϕ in context Γ

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

and this definition is extended to non-normal forms in the following way

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

Normalization of SSF$^\omega$

Lemma (Substitution for the Interpretation of Types)

If $n' \in [\![\phi']\!]_{\Gamma,x:\phi,\Gamma'}$, $n \in [\![\phi]\!]_\Gamma$, then $[n/x]^\phi n' \in [\![\phi']\!]_{\Gamma,\Gamma'}$.

Theorem (Type Soundness Normal Types)

If $\Gamma \vdash t : \phi$ and ϕ is normal then $t \in [\![\phi]\!]_\Gamma$.

Lemma (Preservation of Types for Kinding)

i. *If $(\Gamma, x : \phi, \Gamma')$ Ok and $\phi \rightsquigarrow \phi'$ then $(\Gamma, x : \phi', \Gamma')$ Ok.*
ii. *If $\Gamma \vdash \phi : K$ and $\phi \rightsquigarrow \phi'$ then there exists a Γ' such that $\Gamma' \vdash \phi' : K$.*

Lemma (Preservation of Types for Typing)

If $\Gamma \vdash t : \phi$ and $\phi \rightsquigarrow \phi'$ then there exists a Γ' such that $\Gamma' \vdash t : \phi'$.

Normalization of SSF$^\omega$

Theorem (Type Soundness)

If $\Gamma \vdash t : \phi$ then $\phi \leadsto^! \psi$, and there exists a Γ' such that $t \in [\![\psi]\!]_{\Gamma'}$.

Proof.

By regularity we know $\Gamma \vdash \phi : K$ for some kind K and by normalization of the type level there exists a normal type ψ such that $\phi \leadsto^! \psi$. Finally, by preservation of types for typing there exists a Γ' such that $\Gamma' \vdash t : \psi$. Thus, by type soundness of normal types $t \in [\![\psi]\!]_{\Gamma'}$. □

Concluding remarks

- We have analyzed several systems.
  - Simply Typed λ-Calculus (STLC)
  - Simply Typed λ-Calculus$^{=}$
    - An extension of STLC with a primitive notion of equality between types.
  - Stratified System F (SSF)
  - Stratified System F^{+}
    - An extension of SSF with sum types and commuting conversions.
  - Dependent Stratified System F
    - An extension of SSF with dependent function types and a primitive notion of equality between terms.
  - Stratified System F$^{\omega}$
    - An extension of SSF with type-level computation.
- Future work.
  - Extend to higher ordinals. Goal: System T.
  - Look into full System F and type theories with control.
- Thank you all of you for listening.