Exploring the Reach of Hereditary Substitution

Harley Eades and Aaron Stump

Computer Science
The University of Iowa

TYPES 2011
Introduction

- Tait-Girard’s reducibility is the most often used proof technique for proving normalization.
  - Complex.
    - Type soundness theorem requires universal quantification over all substitutions.
  - Requires mutual recursion.
- Hereditary substitution shows promise of being less complex than reducibility.
  - No universal quantification needed in the statement of the type soundness theorem.
  - In general not dependent on mutual recursion.
  - One major draw back: we are unsure what systems hereditary substitution can be applied to.
    - This is the focus of our work.
Introduction

- Stratified System $F^+$. 

- The hereditary substitution function.
  - Well-founded ordering on types.
  - Properties of the hereditary substitution function.

- Concluding normalization.
  - The interpretation of types.
  - Substitution for the interpretation of types.
  - Type soundness.
Stratified System F$^+$ (SSF$^+$)

- SSF$^+$ is an extension of the system Stratified System F first analyzed by D. Leivant and N. Danner.

- Syntax for kinds, types, and terms:

  
  \begin{align*}
  K &:= *_0 \mid *_1 \mid \ldots \\
  \phi &:= X \mid \phi \rightarrow \phi \mid \forall X : K.\phi \mid \phi + \phi \\
  t &:= x \mid \lambda x : \phi. t \mid t t \mid \Lambda X : K.t \mid t[\phi] \mid inl(t) \mid inr(t) \mid \text{case } t \text{ of } x.t, x.t
  \end{align*}
Stratified System $F^+$ (SSF$^+$)

- **Kind assignment rules:**

  \[
  \Gamma \vdash \phi_1 : *_p \quad \Gamma \vdash \phi_2 : *_q \\
  \Gamma \vdash \phi_1 \rightarrow \phi_2 : *_{\max(p,q)}
  \]

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

  \[
  \Gamma \vdash \phi_1 : *_p \quad \Gamma \vdash \phi_2 : *_q \\
  \Gamma \vdash \phi_1 + \phi_2 : *_{\max(p,q)}
  \]

  \[
  \Gamma(X) = *_p \\
  \Gamma \text{Ok} \quad p \leq q \\
  \Gamma \vdash X : *_q
  \]
Stratified System $F^+$ (SSF$^+$)

The type assignment rules:

1. \[ \Gamma(x) = \phi \]
2. \[ \Gamma \quad \text{Ok} \]
3. \[ \Gamma \vdash x : \phi \]

1. \[ \Gamma, x : \phi_1 \vdash t : \phi_2 \]
2. \[ \Gamma \vdash \lambda x : \phi_1.t : \phi_1 \rightarrow \phi_2 \]
3. \[ \Gamma \vdash t_2 : \phi_1 \]
4. \[ \Gamma \vdash t_1 t_2 : \phi_2 \]

1. \[ \Gamma, X : \ast_l \vdash t : \phi \]
2. \[ \Gamma \vdash \forall X : \ast_l.\phi \]
3. \[ \Gamma \vdash t[\phi_2] : [\phi_2/X] \phi_1 \]
4. \[ \Gamma \vdash \text{case } t \text{ of } x.t_1, x.t_2 : \psi \]

Harley Eades
Exploring the Reach of Hereditary Substitution
Stratified System $F^+$ (SSF$^+$)

The reduction rules:

$$(\Lambda X : \ast p.t)[\phi] \Rightarrow [\phi/X]t$$

$$(\lambda x : \phi.t)t' \Rightarrow [t'/x]t$$

case $inl(t)$ of $x.t_1,x.t_2 \Rightarrow [t/x]t_1$

case $inr(t)$ of $x.t_1,x.t_2 \Rightarrow [t/x]t_2$

Commuting Conversions:

$$(\text{case } t \text{ of } x.t_1, x.t_2) t' \Rightarrow \text{case } t \text{ of } x.(t_1 t'), x.(t_2 t')$$

$$(\text{case } t \text{ of } x.t_1, x.t_2)[\phi] \Rightarrow (\text{case } t \text{ of } x.(t_1[\phi]), x.(t_2[\phi]))$$

case (case $t$ of $x.t_1, x.t_2$) of $y.s_1, y.s_2$

$\Rightarrow \text{case } t \text{ of } x.(\text{case } t_1 \text{ of } y.s_1, y.s_2), x.(\text{case } t_1 \text{ of } y.s_1, y.s_2)$$
Stratified System $F^+$ (SSF$^+$)

- The reduction rules:

- Redex
  \[(\Lambda X : *p.t)[\phi] \rightsquigarrow [\phi/X]t\]
  \[(\lambda x : \phi.t)t' \rightsquigarrow [t'/x]t\]
  case $inl(t)$ of $x.t_1, x.t_2$ $\rightsquigarrow [t/x]t_1$
  case $inr(t)$ of $x.t_1, x.t_2$ $\rightsquigarrow [t/x]t_2$

- Commuting Conversions:

- Structural redex
  \[(\text{case } t \text{ of } x.t_1, x.t_2) t' \rightsquigarrow \text{case } t \text{ of } x(t_1 t'), x(t_2 t')\]

  \[(\text{case } t \text{ of } x.t_1, x.t_2)[\phi] \rightsquigarrow (\text{case } t \text{ of } x(t_1[\phi]), x(t_2[\phi]))\]

  case (case $t$ of $x.t_1, x.t_2$) of $y.s_1, y.s_2$ $\rightsquigarrow \text{case } t \text{ of } x(\text{case } t_1 \text{ of } y.s_1, y.s_2),
  x(\text{case } t_1 \text{ of } y.s_1, y.s_2)\]
Well-founded ordering on types

Definition (well-founded ordering on types)

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

$$
\phi_1 \rightarrow \phi_2 \succ_\Gamma \phi_1 \\
\phi_1 \rightarrow \phi_2 \succ_\Gamma \phi_2 \\
\phi_1 + \phi_2 \succ_\Gamma \phi_1 \\
\phi_1 + \phi_2 \succ_\Gamma \phi_2 \\
\forall X : \ast_l. \phi \succ_\Gamma [\phi'/X]\phi \text{ where } \Gamma \vdash \phi' : \ast_l.
$$

Theorem ($\succ_\Gamma$ is well-founded)

*The ordering $\succ_\Gamma$ is well-founded on types $\phi$ such that $\Gamma \vdash \phi : \ast_l$ for some $l$.***
Syntax: $[t/x]^φ t' = t''$.

Like ordinary capture avoiding substitution.

Except, if the substitution introduces a redex, then that redex is recursively reduced.

Example: $((\lambda z : b.z)/x)^b x y) \rightsquigarrow (\lambda z : b.z)y \rightsquigarrow [y/z]^b z = y$. 

Exploring the Reach of Hereditary Substitution
The hereditary substitution function for SSF$^+$

\[
\text{ctype}_\phi(x, x) = \phi
\]

\[
\text{ctype}_\phi(x, t_1, t_2) = \phi''
\]
Where \( \text{ctype}_\phi(x, t_1) = \phi' \rightarrow \phi'' \).

\[
\text{ctype}_\phi(x, t[\phi']) = [\phi'/X]\phi''
\]
Where \( \text{ctype}_\phi(x, t) = \forall X : \ast_1.\phi'' \).

**Lemma (Properties of \( \text{ctype}_\phi \))**

If \( \Gamma, x : \phi, \Gamma' \vdash t : \phi' \) and \( \text{ctype}_\phi(x, t) = \phi'' \) then \( \text{head}(t) = x \), \( \phi' \equiv \phi'' \), and \( \phi' \leq_\Gamma \phi \).
The hereditary substitution function for SSF\(^+\)

\[
\text{app}_\phi \ t_1 \ t_2 = t_1 \ t_2
\]

Where \(t_1\) is not a \(\lambda\)-abstraction or a case construct.

\[
\text{app}_\phi (\lambda x : \phi'.t_1) \ t_2 = [t_2/x]^{\phi'} t_1
\]

\[
\text{app}_\phi (\text{case } t_0 \text{ of } x.t_1, x.t_2) \ t = \text{case } t_0 \text{ of } x.(\text{app}_\phi t_1 \ t), x.(\text{app}_\phi t_2 \ t)
\]

\[
\text{rcase}_\phi \ t_0 \ y \ t_1 \ t_2 = \text{case } t_0 \text{ of } y.t_1, y.t_2
\]

Where \(t_0\) is not an inject-left or an inject-right term or a case construct.

\[
\text{rcase}_\phi \ \text{inl}(t') \ y \ t_1 \ t_2 = [t'/y]^{\phi_1} t_1
\]

\[
\text{rcase}_\phi \ \text{inr}(t') \ y \ t_1 \ t_2 = [t'/y]^{\phi_2} t_2
\]

\[
\text{rcase}_\phi (\text{case } t'_0 \text{ of } x.t'_1, x.t'_2) \ y \ t_1 \ t_2 = \text{case } t'_0 \text{ of } x.(\text{rcase}_\phi t'_1 \ y \ t_1 \ t_2), x.(\text{rcase}_\phi t'_2 \ y \ t_1 \ t_2)
\]
\[ [t/x]^{\phi} x = t \]

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

Where \( y \) is a variable distinct from \( x \).

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

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

\[ [t/x]^{\phi} inr(t') = inr([t/x]^{\phi} t') \]

\[ [t/x]^{\phi} inl(t') = inl([t/x]^{\phi} t') \]
\[ [t/x]^{\phi}(t_1 \ t_2) = ([t/x]^{\phi}t_1) ([t/x]^{\phi}t_2) \]
Where \([t/x]^{\phi}t_1\) is not a \(\lambda\)-abstraction or a case construct, or both \([t/x]^{\phi}t_1\) and \(t_1\) are \(\lambda\)-abstractions or case constructs, or \(\text{ctype}_{\phi}(x, t_1)\) 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 \(\phi''\) and \(\text{ctype}_{\phi}(x, t_1) = \phi'' \rightarrow \phi'.\)

\[ [t/x]^{\phi}(t_1 \ t_2) = \text{case } w \text{ of } y.(\text{app}_\phi \ r ([t/x]^{\phi}t_2)), y.(\text{app}_\phi \ s ([t/x]^{\phi}t_2)) \]
Where \([t/x]^{\phi}t_1\) \(\equiv \text{case } w \text{ of } y.r,y.s\) for some terms \(w, r, s\) and variable \(y, \) and \(\text{ctype}_{\phi}(x, t_1) = \phi'' \rightarrow \phi'.\)

\[ [t/x]^{\phi}(t'[\phi']) = ([t/x]^{\phi}t')[\phi'] \]
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 \(\text{ctype}_{\phi}(x, t') = \forall X : *_l.\phi''.\)
\[ [t/x]^\phi (\text{case } t_0 \text{ of } y.t_1,y.t_2) = \]
\text{case } ([t/x]^\phi t_0) \text{ of } y.([t/x]^\phi t_1),y.([t/x]^\phi t_2) \]

Where \([t/x]^\phi t_0\) is not an inject-left or an inject-right term or a case construct, or \([t/x]^\phi t_0\) and \(t_0\) are both inject-left or inject-right terms or case constructs, or \(c_{type}^\phi (x, t_0)\) is undefined.

\[ [t/x]^\phi (\text{case } t_0 \text{ of } y.t_1,y.t_2) = \]
\text{rcase}^\phi ([t/x]^\phi t_0) y ([t/x]^\phi t_1) ([t/x]^\phi t_2) \]

Where \([t/x]^\phi t_0\) is an inject-left or an inject-right term or a case construct and \(c_{type}^\phi (x, t_0) = \phi_1 + \phi_2.\)
The $\text{ctype}_\phi$ properties

**Lemma (Properties of $\text{ctype}_\phi$)**

1. 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 $\psi$ such that $\text{ctype}_{\phi}(x, t_1) = \psi$.

2. If $\Gamma, x : \phi, \Gamma' \vdash t_1 t_2 : \phi', \Gamma \vdash t : \phi$, $[t/x]^{\phi}t_1 = \text{case } t'_0 \text{ of } y.t'_1, y.t'_2$, and $t_1$ is not then there exists a type $\psi$ such that $\text{ctype}_{\phi}(x, t_1) = \psi$.

3. 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 $\psi$ such that $\text{ctype}_{\phi}(x, t') = \psi$.

4. If $\Gamma, x : \phi, \Gamma' \vdash \text{case } t_0 \text{ of } y.t_1,y.t_2 : \phi'$, $\Gamma \vdash t : \phi$, $[t/x]^{\phi}t_0 = \text{case } t'_0 \text{ of } z.t'_1,z.t'_2$, and $t_0$ is not then there exists a type $\psi$ such that $\text{ctype}_{\phi}(x, t_0) = \psi$.

5. If $\Gamma, x : \phi, \Gamma' \vdash \text{case } t_0 \text{ of } y.t_1,y.t_2 : \phi'$, $\Gamma \vdash t : \phi$, $[t/x]^{\phi}t_0 = \text{inl}(t')$, and $t_0$ is not then there exists a type $\psi$ such that $\text{ctype}_{\phi}(x, t_0) = \psi$.

6. If $\Gamma, x : \phi, \Gamma' \vdash \text{case } t_0 \text{ of } y.t_1,y.t_2 : \phi'$, $\Gamma \vdash t : \phi$, $[t/x]^{\phi}t_0 = \text{inr}(t')$, and $t_0$ is not then there exists a type $\psi$ such that $\text{ctype}_{\phi}(x, t_0) = \psi$. 
Properties of the hereditary substitution function

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 (Redex Preserving)

If $\Gamma \vdash t : \phi$, $\Gamma, x : \phi, \Gamma' \vdash t' : \phi'$ then $|rset(t', t)| \geq |rset([t/x]^\phi t')|$. 
Examples: rset and commuting conversions

- Structural redexes are not preserved by the hereditary substitution function in general.

Let
\[ t \equiv \text{inl}(a), \text{such that} \ a : \phi_1 \vdash t : \phi_1 + \phi_2 \text{ and} \]
\[ t' \equiv \text{case (case x of z.z,z.z) of y.y,y.y}. \]

So
\[ [t/x]^{\phi_1 + \phi_2} t' = \]
\[ \text{case ([t/x]^{\phi_1 + \phi_2}(\text{case x of z.z,z.z})) of y.([t/x]^{\phi_1 + \phi_2} y),y.([t/x]^{\phi_1 + \phi_2} y)}. \]

Now
\[ [t/x]^{\phi_1 + \phi_2} (\text{case x of z.z,z.z}) = \]
\[ \text{rcase}^{\phi_1 + \phi_2} [t/x]^{\phi_1 + \phi_2} x [t/x]^{\phi_1 + \phi_2} z [t/x]^{\phi_1 + \phi_2} z, \]

because
\[ [t/x]^{\phi_1 + \phi_2} x = \text{inl}(a), \text{x is not an inject-left term, and} \]
\[ \text{ctype}^{\phi_1 + \phi_2} (x, x) = \phi_1 + \phi_2. \]

Finally,
\[ [t/x]^{\phi_1 + \phi_2} (\text{case x of z.z,z.z}) = [a/z]^{\phi_1} z = a, \text{which implies,} \]
\[ [t/x]^{\phi_1 + \phi_2} t' = \text{case a of y.y,y.y}. \]
Properties of the hereditary substitution function

**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 $[t/x]^{\phi}t' \leadsto^* [t/x]^{\phi}t'$. 
Concluding normalization

**Definition**

\[ n \in [\phi]_{\Gamma} \iff \Gamma \vdash n : \phi. \]

**Lemma (Substitution for the Interpretation of Types)**

*If* \( n' \in [\phi']_{\Gamma, x: \phi, \Gamma'} \), \( n \in [\phi]_{\Gamma} \), *then* \([n/x]_{\phi} n' \in [\phi']_{\Gamma, \Gamma'} \).*

**Proof.**

By Totality we know there exists a term \( \hat{n} \) such that \([n/x]_{\phi} n' = \hat{n} \) and \( \Gamma, \Gamma' \vdash \hat{n} : \phi' \) and by Normality Preservation \( \hat{n} \) is normal. Therefore, \([n/x]_{\phi} n' = \hat{n} \in [\phi']_{\Gamma, \Gamma'} \).*
Concluding normalization

**Theorem (Type Soundness)**

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

**Corollary (Normalization)**

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

- Thank you all of you for listening.