Type Systems

Lecture 5: System F and Church Encodings

Neel Krishnaswami
University of Cambridge

System F, The Girard-Reynolds Polymorphic Lambda Calculus

Types	A	$::=$	$\alpha \mid A \to B \mid \forall\alpha.\, A$
Terms	e	$::=$	$x \mid \lambda x : A.\, e \mid e\, e \mid \Lambda\alpha.\, e \mid e\, A$
Type Contexts	Θ	$::=$	$\cdot \mid \Theta, \alpha$
Term Contexts	Γ	$::=$	$\cdot \mid \Gamma, x : A$

Judgement	Notation
Well-formedness of types	$\Theta \vdash A$ type
Well-formedness of term contexts	$\Theta \vdash \Gamma$ ctx
Term typing	$\Theta \vdash \Gamma : eA$

Well-formedness of Types

$$\frac{\alpha \in \Theta}{\Theta \vdash \alpha \text{ type}} \qquad \frac{\Theta \vdash A \text{ type} \qquad \Theta \vdash B \text{ type}}{\Theta \vdash A \to B \text{ type}}$$

$$\frac{\Theta, \alpha \vdash A \text{ type}}{\Theta \vdash \forall \alpha. A \text{ type}}$$

- Judgement $\Theta \vdash A$ type checks if a type is well-formed
- Because types can have free variables, we need to check if a type is well-scoped

Well-formedness of Term Contexts

$$\text{Term Variable Contexts} \quad \Gamma \quad ::= \quad \cdot \mid \Gamma, x : A$$

$$\frac{}{\Theta \vdash \cdot \ \mathsf{ctx}} \qquad\qquad \frac{\Theta \vdash \Gamma \ \mathsf{ctx} \qquad \Theta \vdash A \ \mathsf{type}}{\Theta \vdash \Gamma, x : A \ \mathsf{ctx}}$$

- Judgement $\Theta \vdash \Gamma$ type checks if a *term context* is well-formed
- We need this because contexts associate variables with types, and types now have a well-formedness condition

Typing for System F

$$\frac{x : A \in \Gamma}{\Theta; \Gamma \vdash x : A}$$

$$\frac{\Theta \vdash A \text{ type} \qquad \Theta; \Gamma, x : A \vdash e : B}{\Theta; \Gamma \vdash \lambda x : A.\, e : A \to B}$$

$$\frac{\Theta; \Gamma \vdash e : A \to B \qquad \Theta; \Gamma \vdash e' : A}{\Theta; \Gamma \vdash e\, e' : B}$$

$$\frac{\Theta, \alpha; \Gamma \vdash e : B}{\Theta; \Gamma \vdash \Lambda\alpha.\, e : \forall\alpha.\, B} \qquad \frac{\Theta; \Gamma \vdash e : \forall\alpha.\, B \qquad \Theta \vdash A \text{ type}}{\Theta; \Gamma \vdash e\, A : \boxed{[A/\alpha]B}}$$

- Note the presence of substitution in the typing rules!

Operational Semantics

$$\text{Values} \quad v \quad ::= \quad \lambda x : A.\, e \mid \Lambda\alpha.\, e$$

$$\frac{e_0 \leadsto e_0'}{e_0\, e_1 \leadsto e_0'\, e_1}\ \textsc{CongFun} \qquad \frac{e_1 \leadsto e_1'}{v_0\, e_1 \leadsto v_0\, e_1'}\ \textsc{CongFunArg}$$

$$\frac{}{(\lambda x : A.\, e)\, v \leadsto [v/x]e}\ \textsc{FunEval}$$

$$\frac{e \leadsto e'}{e\, A \leadsto e'\, A}\ \textsc{CongForall} \qquad \frac{}{(\Lambda\alpha.\, e)\, A \leadsto [A/\alpha]e}\ \textsc{ForallEval}$$

The Bookkeeping

- Ultimately, we want to prove type safety for System F
- However, the introduction of type variables means that a fair amount of additional administrative overhead is introduced
- This may look intimidating on first glance, BUT really it's all just about keeping track of the free variables in types
- As a result, none of these lemmas are hard – just a little tedious

Structural Properties and Substitution for Types

1. (Type Weakening) If $\Theta, \Theta' \vdash A$ type then $\Theta, \beta, \Theta' \vdash A$ type.
2. (Type Exchange) If $\Theta, \beta, \gamma, \Theta' \vdash A$ type then $\Theta, \gamma, \beta, \Theta' \vdash A$ type
3. (Type Substitution) If $\Theta \vdash A$ type and $\Theta, \alpha \vdash B$ type then $\Theta \vdash [A/\alpha]B$ type

- These follow the pattern in lecture 1, except with fewer cases
- Needed to handle the type application rule

Structural Properties and Substitutions for Contexts

1. (Context Weakening) If $\Theta, \Theta' \vdash \Gamma$ ctx then $\Theta, \alpha, \Theta' \vdash \Gamma$ ctx
2. (Context Exchange) If $\Theta, \beta, \gamma, \Theta' \vdash \Gamma$ ctx then $\Theta, \gamma, \beta, \Theta' \vdash \Gamma$ ctx
3. (Context Substitution) If $\Theta \vdash A$ type and $\Theta, \alpha \vdash \Gamma$ type then $\Theta \vdash [A/\alpha]\Gamma$ type

- This just lifts the type-level structural properties to contexts
- Proof via induction on derivations of $\Theta \vdash \Gamma$ ctx

Regularity of Typing

Regularity: If $\Theta \vdash \Gamma$ ctx and $\Theta; \Gamma \vdash e : A$ then $\Theta \vdash A$ type

Proof: By induction on the derivation of $\Theta; \Gamma \vdash e : A$

- This just says if typechecking succeeds, then it found a well-formed type

Structural Properties and Substitution of Types into Terms

- (Type Weakening of Terms) If $\Theta, \Theta' \vdash \Gamma$ ctx and $\Theta, \Theta'; \Gamma \vdash e : A$ then $\Theta, \alpha, \Theta'; \Gamma \vdash e : A$.
- (Type Exchange of Terms) If $\Theta, \alpha, \beta, \Theta' \vdash \Gamma$ ctx and $\Theta, \alpha, \beta, \Theta'; \Gamma \vdash e : A$ then $\Theta, \beta, \alpha, \Theta'; \Gamma \vdash e : A$.
- (Type Substitution of Terms) If $\Theta, \alpha \vdash \Gamma$ ctx and $\Theta \vdash A$ type and $\Theta, \alpha; \Gamma \vdash e : B$ then $\Theta; [A/\alpha]\Gamma \vdash [A/\alpha]e : [A/\alpha]B$.

Structural Properties and Substitution for Term Variables

- (Weakening for Terms) If $\Theta \vdash \Gamma, \Gamma'$ ctx and $\Theta \vdash B$ type and $\Theta; \Gamma, \Gamma' \vdash e : A$ then $\Theta; \Gamma, y : B, \Gamma' \vdash e : A$
- (Exchange for Terms) If $\Theta \vdash \Gamma, y : B, z : C, \Gamma'$ ctx and $\Theta; \Gamma, y : B, z : C, \Gamma' \vdash e : A$, then $\Theta; \Gamma, z : C, y : B, \Gamma' \vdash e : A$
- (Substitution of Terms) If $\Theta \vdash \Gamma, x : A$ ctx and $\Theta; \Gamma \vdash e : A$ and $\Theta; \Gamma, x : A \vdash e' : B$ then $\Theta; \Gamma \vdash [e/x]e' : B$.

Summary

- There are two sets of substitution theorems, since there are two contexts
- We also need to assume well-formedness conditions
- But proofs are all otherwise similar to the simply-typed case

Type Safety

Progress: If $\cdot; \cdot \vdash e : A$ then either e is a value or $e \leadsto e'$.

Type preservation: If $\cdot; \cdot \vdash e : A$ and $e \leadsto e'$ then $\cdot; \cdot \vdash e' : A$.

Progress: Big Lambdas

Proof by induction on derivations:

$$\dfrac{\overbrace{\cdot;\cdot \vdash e : \forall\alpha.\,B}^{(2)} \qquad \overbrace{\cdot \vdash A\ \text{type}}^{(3)}}{\cdot;\cdot \vdash e\,A : [A/\alpha]B}$$

(1)	$\cdot;\cdot \vdash e\,A : [A/\alpha]B$	Assumption
(4)	$e \leadsto e'$ or e is a value	Induction on (2)
	Case on (4)	
(5)	Case $e \leadsto e'$:	
(6)	$e\,A \leadsto e'\,A$	by CongForall on (5)
(7)	Case e is a value:	
(8)	$e = \Lambda\alpha.\,e'$	By canonical forms on (2)
(9)	$(\Lambda\alpha.\,e')\,A \leadsto [A/\alpha]e$	By ForallEval

By induction on the derivation of $e \leadsto e'$:

(1) $$\dfrac{}{(\Lambda\alpha.\, e)\, A \leadsto [A/\alpha]e}\ \textsc{ForallEval}$$ Assumption

(2) $$\dfrac{\dfrac{\overbrace{\alpha; \cdot \vdash e : B}^{(3)}}{\cdot; \cdot \vdash \Lambda\alpha.\, e : \forall\alpha.\, B} \qquad \overbrace{\cdot \vdash A \text{ type}}^{(4)}}{\cdot; \cdot \vdash (\Lambda\alpha.\, e)\, A : [A/\alpha]B}$$ Assumption

(5) $\cdot; \cdot \vdash [A/\alpha]e : [A/\alpha]B$ Type subst. on (3), (4)

Church Encodings: Representing Data with Functions

- System has the types $\forall\alpha. A$ and $A \to B$
- No booleans, sums, numbers, tuples or anything else
- Seemingly, there is no data in this calculus
- Surprisingly, it is unnecessary!
- Discovered in 1941 by Alonzo Church
- The idea:
    1. Data is used to make choices
    2. Based on the choice, you perform different results
    3. So we can encode data as functions which take different possible results, and return the right one

Church Encodings: Booleans

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{bool}} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{bool}}$$

$$\frac{\Gamma \vdash e : \mathsf{bool} \qquad \Gamma \vdash e' : X \qquad \Gamma \vdash e'' : X}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e'\ \mathsf{else}\ e'' : X}$$

- Boolean type has two values, true and false
- Conditional switches between two X's based on e's value

Type		Encoding
bool	$\triangleq$	$\forall\alpha.\, \alpha \to \alpha \to \alpha$
True	$\triangleq$	$\Lambda\alpha.\, \lambda x : \alpha.\, \lambda y : \alpha.\, x$
False	$\triangleq$	$\Lambda\alpha.\, \lambda x : \alpha.\, \lambda y : \alpha.\, y$
if e then e' else e'' : X	$\triangleq$	$e\, X\, e'\, e''$

Evaluating Church conditionals

$$
\begin{aligned}
\text{if true then } e' \text{ else } e'' : A &= \text{true } A\ e'\ e'' \\
&= (\Lambda\alpha.\ \lambda x : \alpha.\ \lambda y : \alpha.\ x)\ A\ e'\ e'' \\
&= (\lambda x : A.\ \lambda y : A.\ x)\ e'\ e'' \\
&= (\lambda y : A.\ e')\ e'' \\
&= e'
\end{aligned}
$$

$$
\begin{aligned}
\text{if false then } e' \text{ else } e'' : A &= \text{false } A\ e'\ e'' \\
&= (\Lambda\alpha.\ \lambda x : \alpha.\ \lambda y : \alpha.\ y)\ A\ e'\ e'' \\
&= (\lambda x : A.\ \lambda y : A.\ y)\ e'\ e'' \\
&= (\lambda y : A.\ y)\ e'' \\
&= e''
\end{aligned}
$$

Church Encodings: Pairs

Type		Encoding
$X \times Y$	$\triangleq$	$\forall\alpha.\,(X \to Y \to \alpha) \to \alpha$
$\langle e, e' \rangle$	$\triangleq$	$\Lambda\alpha.\,\lambda k : X \to Y \to \alpha.\, k\, e\, e'$
$\mathsf{fst}\, e$	$\triangleq$	$e\, X\, (\lambda x : X.\, \lambda y : Y.\, x)$
$\mathsf{snd}\, e$	$\triangleq$	$e\, Y\, (\lambda x : X.\, \lambda y : Y.\, y)$

Evaluating Church Pairs

$$
\begin{array}{lcl}
\mathsf{fst}\,\langle e, e'\rangle & = & \langle e, e'\rangle \; X\,(\lambda x : X.\,\lambda y : Y.\,x) \\
& = & (\Lambda\alpha.\,\lambda k : X \to Y \to \alpha.\,k\,e\,e')\,X\,(\lambda x : X.\,\lambda y : Y.\,x) \\
& = & (\lambda k : X \to Y \to X.\,k\,e\,e')\,(\lambda x : X.\,\lambda y : Y.\,x) \\
& = & (\lambda x : X.\,\lambda y : Y.\,x)\,e\,e' \\
& = & (\lambda y : Y.\,e)\,e' \\
& = & e
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{snd}\,\langle e, e'\rangle & = & \langle e, e'\rangle \; Y\,(\lambda x : X.\,\lambda y : Y.\,y) \\
& = & (\Lambda\alpha.\,\lambda k : X \to Y \to \alpha.\,k\,e\,e')\,Y\,(\lambda x : X.\,\lambda y : Y.\,y) \\
& = & (\lambda k : X \to Y \to Y.\,k\,e\,e')\,(\lambda x : X.\,\lambda y : Y.\,y) \\
& = & (\lambda x : X.\,\lambda y : Y.\,y)\,e\,e' \\
& = & (\lambda y : Y.\,y)\,e' \\
& = & e'
\end{array}
$$

Church Encodings: Sums

Type	Encoding
$X + Y$	$\forall \alpha.\, (X \to \alpha) \to (Y \to \alpha) \to \alpha$
$\mathsf{L}\, e$	$\Lambda \alpha.\, \lambda f : X \to \alpha.\, \lambda g : Y \to \alpha.\, f\, e$
$\mathsf{R}\, e$	$\Lambda \alpha.\, \lambda f : X \to \alpha.\, \lambda g : Y \to \alpha.\, g\, e$
$\mathsf{case}(e, \mathsf{L}\, x \to e_1, \mathsf{R}\, y \to e_2) : Z$	$e\, Z\, (\lambda x : X \to Z.\, e_1)\ (\lambda y : Y \to Z.\, e_2)$

Evaluating Church Sums

$$
\begin{aligned}
&\mathrm{case}(\mathsf{L}\, e, \mathsf{L}\, x \to e_1, \mathsf{R}\, y \to e_2) : Z \\
&= (\mathsf{L}\, e)\, Z\, (\lambda x : X \to Z.\, e_1)\; (\lambda y : Y \to Z.\, e_2) \\
&= (\Lambda\alpha.\, \lambda f : X \to \alpha.\, \lambda g : Y \to \alpha.\, f\, e) \\
&\qquad Z\, (\lambda x : X \to Z.\, e_1)\; (\lambda y : Y \to Z.\, e_2) \\
&= (\lambda f : X \to Z.\, \lambda g : Y \to Z.\, f\, e) \\
&\qquad (\lambda x : X \to Z.\, e_1)\; (\lambda y : Y \to Z.\, e_2) \\
&= (\lambda g : Y \to Z.\, (\lambda x : X \to Z.\, e_1)\, e) \\
&\qquad (\lambda y : Y \to Z.\, e_2) \\
&= (\lambda x : X \to Z.\, e_1)\, e \\
&= [e/x]e_1
\end{aligned}
$$

Church Encodings: Natural Numbers

Type	Encoding
$\mathbb{N}$	$\forall\alpha.\, \alpha \to (\alpha \to \alpha) \to \alpha$
z	$\Lambda\alpha.\, \lambda z : \alpha.\, \lambda s : \alpha \to \alpha.\, z$
$\mathsf{s}(e)$	$\Lambda\alpha.\, \lambda z : \alpha.\, \lambda s : \alpha \to \alpha.\, s\ (e\ \alpha\ z\ s)$
$\mathsf{iter}(e, \mathsf{z} \to e_z, \mathsf{s}(x) \to e_s) : X$	$e\, X\, e_z\, (\lambda x : X.\, e_s)$

Evaluating Church Naturals

$$
\begin{aligned}
&\mathsf{iter}(\mathsf{z}, \mathsf{z} \to e_z, \mathsf{s}(x) \to e_s) \\
&\quad = \mathsf{z}\, X\, e_z\, (\lambda x : X.\, e_s) \\
&\quad = (\Lambda\alpha.\, \lambda z : \alpha.\, \lambda s : \alpha \to \alpha.\, z)\, X\, e_z\, (\lambda x : X.\, e_s) \\
&\quad = (\lambda z : X.\, \lambda s : X \to X.\, z)\, e_z\, (\lambda x : X.\, e_s) \\
&\quad = (\lambda s : X \to X.\, e_z)\, (\lambda x : X.\, e_s) \\
&\quad = e_z
\end{aligned}
$$

Evaluating Church Naturals

$$
\begin{aligned}
&\mathsf{iter}(\mathsf{s}(e), \mathsf{z} \to e_z, \mathsf{s}(x) \to e_s) \\
&\quad = (\mathsf{s}(e))\, X\, e_z\, (\lambda x : X.\, e_s) \\
&\quad = (\Lambda\alpha.\, \lambda z : \alpha.\, \lambda s : \alpha \to \alpha.\, s\ (e\ \alpha\ z\ s))\, X\, e_z\, (\lambda x : X.\, e_s) \\
&\quad = (\lambda z : X.\, \lambda s : X \to X.\, s\ (e\ X\ z\ s))\, e_z\, (\lambda x : X.\, e_s) \\
&\quad = (\lambda s : X \to X.\, s\ (e\ X\ e_z\ s))\, (\lambda x : X.\, e_s) \\
&\quad = (\lambda x : X.\, e_s)\ (e\ X\ e_z\ (\lambda x : X.\, e_s)) \\
&\quad = (\lambda x : X.\, e_s)\ \mathsf{iter}(e, \mathsf{z} \to e_z, \mathsf{s}(x) \to e_s) \\
&\quad = [\mathsf{iter}(e, \mathsf{z} \to e_z, \mathsf{s}(x) \to e_s)/x]e_s
\end{aligned}
$$

Church Encodings: Lists

Type	Encoding
$\mathrm{list}\, X$	$\forall\alpha.\, \alpha \to (X \to \alpha \to \alpha) \to \alpha$
$[]$	$\Lambda\alpha.\, \lambda n : \alpha.\, \lambda c : X \to \alpha \to \alpha.\, n$
$e :: e'$	$\Lambda\alpha.\, \lambda n : \alpha.\, \lambda c : X \to \alpha \to \alpha.\, c\; e\; (e'\; \alpha\; n\; c)$

$$\mathrm{fold}(e, [] \to e_n, x :: r \to e_c) : Z = e\; Z\; e_n\; (\lambda x : X.\, \lambda r : Z.\, e_c)$$

Conclusions

- System F is very simple, and very expressive
- Formal basis of polymorphism in ML, Java, Haskell, etc.
- Surprise: from polymorphism and functions, data is definable

Exercises

1. Prove the regularity lemma.
2. Define a Church encoding for the unit type.
3. Define a Church encoding for the empty type.
4. Define a Church encoding for binary trees, corresponding to the ML datatype
   `type tree = Leaf | Node of tree * X * tree`.