From $\lambda_\to$ to λ_2

TAZMILUR SAAD, Independent, USA

This work documents an autididactic process of extending simply typed lambda calculus to System F.

TERMINOLOGY

We treat typing contexts as sets. $\Gamma, x : \tau$ is referred to as "extending" the context and the extension is sometimes referred to as the new binding.

1 SIMPLY TYPED LAMBDA CALCULUS

1.1 Structural Properties

LEMMA 1.1 (EXCHANGE). *If* $\Gamma_1, x_1 : \tau_1, x_2 : \tau_2, \Gamma_2 \vdash e : \tau$ *then* $\Gamma_1, x_2 : \tau_2, x_1 : \tau_1, \Gamma_2 \vdash e : \tau$.

PROOF. Induction on the typing derivation of $\Gamma_1, x_1 : \tau_1, x_2 : \tau_2, \Gamma_2 \vdash e : \tau$.

[NAT]: This judgement does not depend on the typing context.

[VAR]: $x : \tau$ must be an element of Γ_1 or Γ_2, or it is equal to x_1 or x_2. In each of these cases, the antecedent is satisfied even if we reorder the elements.

[LAM]: Assume the inductive hypothesis. Reordering the extensions of the typing context does not affect the new binding, and we can derive the judgement.

[APP]: Follows from the inductive hypothesis. □

LEMMA 1.2 (WEAKENING). *If* $\Gamma \vdash e : \tau$ *and* $x \notin dom(\Gamma)$*, then* $\Gamma, x : \tau_x \vdash e : \tau$.

PROOF. Induction on the typing derivation of $\Gamma \vdash e : \tau$.

[NAT]: Does not depend on the typing context.

[VAR]: The binding we add to to the context is not within its domain, therefore we can apply the judgement.

[LAM]: Assume the inductive hypothesis. We extend Γ as part of the antecedent — therefore, the judgement follows modulo renaming of further extensions.

[APP]: Follows from the inductive hypothesis. □

LEMMA 1.3 (CONTRACTION). *If* $\Gamma_1, x_2 : \tau_1, x_3 : \tau_1, \Gamma_2 \vdash e : \tau_2$ *then* $\Gamma_1, x_1 : \tau_1, \Gamma_2 \vdash e[x_3/x_1][x_2/x_1] : \tau_2$.

PROOF. QED. □

1.2 Progress & Preservation

LEMMA 1.4 (CANONICAL FORMS). *If* v *is a value of type* `Nat`*, then* $v \in \{0, 1, 2, \dots\}$*. If* v *is a value of type* $\tau \to \tau$*, then* $v = \lambda x : \tau.e$.

PROOF. The metavariable n ranges over the natural numbers, and elements denoted by n are the only elements that can have the type `Nat` according to the typing rules. The LAM rule is the only rule that implies a value has type $\tau \to \tau$, and it asserts that the value is a lambda abstraction. □

NOTE. This can also be proven using the inversion of the typing relation.

Author's address: Tazmilur Saad, Independent, Jersey City, NJ, USA, ssaad@colgate.edu.

$\Gamma ::= \varnothing \mid \Gamma, x : \tau$	typing context
$v ::= \lambda x : \tau.e \mid b$	values
$b ::= n$	base types
$n ::= 0 \mid 1 \mid \ldots$	naturals
$\tau ::= \texttt{Nat} \mid \tau \to \tau$	types
$e ::= x \mid v \mid e\ e$	expressions
$E ::= [\cdot] \mid Ee \mid vE$	evaluation context

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ \textsc{Var} \qquad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1.e : \tau_1 \to \tau_2}\ \textsc{Lam} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2}\ \textsc{App}$$

$$\frac{}{\Gamma \vdash n : \texttt{Nat}}\ \textsc{Nat}$$

Fig. 1. Simply typed lambda calculus

$e ::= \text{let } x = e \text{ in } e \mid \ldots$	expressions
$E ::= \text{let } x = E \text{ in } e \mid \text{let } x = v \text{ in } E \mid \ldots$	evaluation context

$$\text{let } x = v \text{ in } e \to e[v/x] \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2}\ \textsc{Let}$$

Fig. 2. Let

Theorem 1.5 (Progress). *If $\vdash e : \tau$ then e is either a value or there exists an e' such that $e \to e'$.*

Proof. Induction on the typing derivation of $\vdash e : \tau$.

Var. A variable is not well typed in the empty typing context.

Nat, Lam. A natural number and a lambda abstraction are values.

App. Let's assume the inductive hypothesis. If either e_1 or e_2 are not values, then there exists an e' such that a reduction step can be taken. If they are both values, then the canonical forms lemma implies that e_1 is a lambda abstraction since $e_1 : \tau \to \tau$, and we can β-reduce $e_1 e_2$ via $e_1[e_2/x]$.

□

Lemma 1.6 (Preservation of Types under Substitution). *If $\Gamma, x : \tau_1 \vdash e_1 : \tau$ and $\Gamma \vdash e_2 : \tau_1$ then $\Gamma \vdash e_1[e_2/x] : \tau$.*

$$
\begin{aligned}
e &::= () \mid \dots && \text{expressions} \\
v &::= () \mid \dots && \text{values} \\
\tau &::= \texttt{Unit} \mid \dots && \text{types}
\end{aligned}
$$

$$
\frac{}{\Gamma \vdash () : \texttt{Unit}}\ \textsc{Unit}
$$

Fig. 3. Unit

$$
\begin{aligned}
e &::= (e_0, \dots, e_n) \mid e.i \mid \dots && \text{expressions} \\
E &::= (E, \dots) \mid (v, E, \dots) \mid \dots && \text{evaluation context}
\end{aligned}
$$

$$
\frac{\forall i \in \{0, \dots, n\} \quad \Gamma \vdash e_i : \tau_i}{\Gamma \vdash (e_0, \dots, e_n) : (\tau_0, \dots, \tau_n)}\ \textsc{Tuple}
\qquad
\frac{\Gamma \vdash (e_0, \dots, e_n) : (\tau_0, \dots, \tau_n) \quad i \in \{0, \dots, n\}}{\Gamma \vdash e.i : \tau_i}\ \textsc{Proj}
$$

Fig. 4. Tuples

Fig. 5. Records

Fig. 6. Variants

Fig. 7. Fixpoints

Fig. 8. References

Fig. 9. Recursive Types

Fig. 10. Type Reconstruction

2 EXTENSIONS

2.1 Let

2.2 Unit

2.3 Tuples

2.4 Records

2.5 Variants

2.6 Fixpoints

2.7 References

2.8 Recursive Types

3 SYSTEM F

3.1 Type Reconstruction

3.2 Universals

3.3 Existentials

4 ACKNOWLEDGEMENTS

Thanks to Anton Lorenzen for his guidance on the project.

Fig. 11. Universals

Fig. 12. Existentials