

From λ_{\rightarrow} to λ_2

TAZMILUR SAAD, Independent, USA

This work documents an autodidactic 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 $x \notin \text{dom}(\Gamma)$, 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 $\tau \rightarrow \tau$, 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 $\tau \rightarrow \tau$, and it asserts that the value is a lambda abstraction. □

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

$\Gamma ::= \emptyset \mid \Gamma, x : \tau$	typing context
$v ::= \lambda x : \tau. e \mid b$	values
$b ::= n$	base types
$n ::= 0 \mid 1 \mid \dots$	naturals
$\tau ::= \text{Nat} \mid \tau \rightarrow \tau$	types
$e ::= x \mid v \mid e e$	expressions
$E ::= [\cdot] \mid Ee \mid vE$	evaluation context
$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \text{VAR} \quad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2} \text{LAM} \quad \frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2} \text{APP}$ $\frac{}{\Gamma \vdash n : \text{Nat}} \text{NAT}$	

Fig. 1. Simply typed lambda calculus

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

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 $e_1 : \tau \rightarrow \tau$, 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 $\Gamma \vdash e_1[e_2/x] : \tau$.*

PROOF. Induction on the typing derivation of $\Gamma, x : \tau_1 \vdash e_1 : \tau$.

□

2 EXTENSIONS

In each of the following extensions we assume the base system is the simply typed lambda calculus from Fig [?] unless indicated otherwise.

2.1 Let

2.2 Unit

2.3 Tuples

2.4 Records

Let l be a metavariable ranging over labels.

$e ::= \text{let } x = e \text{ in } e \mid \dots$	expressions
$E ::= \text{let } x = E \text{ in } e \mid \text{let } x = v \text{ in } E \mid \dots$	evaluation context
$\text{let } x = v \text{ in } e \rightarrow e[v/x]$	
$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2} \text{LET}$	

Fig. 2. Let

$e ::= () \mid \dots$	expressions
$v ::= () \mid \dots$	values
$\tau ::= \text{Unit} \mid \dots$	types
$\frac{}{\Gamma \vdash () : \text{Unit}} \text{UNIT}$	

Fig. 3. Unit

$v ::= (v_0, \dots, v_n) \mid \dots$	values
$\tau ::= (\tau_0, \dots, \tau_n) \mid \dots$	types
$e ::= (e_0, \dots, e_n) \mid e.i \mid \dots$	expressions
$E ::= (E, \dots) \mid (v, E, \dots) \mid \dots$	evaluation context
$(v_0, \dots, v_n).i \rightarrow v_i \quad \frac{\Gamma \vdash e : (\tau_0, \dots, \tau_n) \quad i \in \{0, \dots, n\}}{\Gamma \vdash e.i : \tau_i} \text{PROJ}$	
$\frac{\forall i \in \{0, \dots, n\} \quad \Gamma \vdash e_i : \tau_i}{\Gamma \vdash (e_0, \dots, e_n) : (\tau_0, \dots, \tau_n)} \text{TUPLE}$	

Fig. 4. Tuples

2.5 Variants

2.6 Fixpoints

2.7 References

We need the simply typed lambda calculus extended with unit types as in Fig 3. l is a metavariable that ranges over memory locations.

$v ::= \{l_0 = v_0, \dots, l_n = v_n\} \mid \dots$	values
$\tau ::= \{l_0 : \tau_0, \dots, l_n : \tau_n\} \mid \dots$	types
$e ::= \{l_0 = e_0, \dots, l_n = e_n\} \mid e.l \mid \dots$	expressions
$E ::= \{l_0 = E, \dots, l_n = e_n\} \mid \{l_0 = v_0, \dots, l_i = e_i, \dots\} \mid \dots$	expression context
$\frac{\{l_0 = v_0, \dots, l_n = v_n\}.l_i \rightarrow v_i \quad e : (l_0 : \tau_0, \dots, l_n : \tau_n) \quad i \in \{0, \dots, n\}}{\Gamma \vdash e.l_i : \tau_i} \text{PROJ}$	
$\frac{\forall i \in \{0, \dots, n\} \quad \Gamma \vdash e_i : \tau_i}{\Gamma \vdash (l_0 = e_0, \dots, l_n = e_n) : (l_0 : \tau_0, \dots, l_n : \tau_n)} \text{TUPLE}$	

Fig. 5. Records

$e ::= \langle l = e \rangle \text{ as } \tau \mid \text{case } e \text{ of } \langle l_i = x_i \rangle \Rightarrow e_i \mid \dots$	expressions
$\tau ::= \langle l_0 : \tau_0, \dots, l_n : \tau_n \rangle \mid \dots$	types
$\frac{\Gamma \vdash e_j : \tau_j}{\Gamma \vdash \langle l_j = e_j \rangle \text{ as } \langle l_0 : \tau_0, \dots, l_n : \tau_n \rangle : \langle l_0 : \tau_0, \dots, l_n : \tau_n \rangle} \text{VARIANT}$	

Fig. 6. Variants

--

Fig. 7. Fixpoints

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.

$e ::= \text{ref } e \mid !e \mid e := e \mid \dots$	expressions
$v ::= l \mid \dots$	values
$\tau ::= \text{Ref } \tau$	types
$\mu ::= \emptyset \mid \mu, l = v$	stores
$\Sigma ::= \emptyset \mid \Sigma, l : \tau$	store typing
$\frac{\Sigma(l) = \tau}{\Gamma \mid \Sigma \vdash l : \text{Ref } \tau} \text{Loc}$	
$l := v \mid \mu \rightarrow () \mid \mu[l \rightarrow v] \quad \frac{\Gamma \mid \Sigma \vdash e_1 : \text{Ref } \tau \quad \Gamma \mid \Sigma \vdash e_2 : \tau}{\Gamma \mid \Sigma \vdash e_1 := e_2 : \text{Unit}} \text{ASSIGN}$	
$\frac{\mu(l) = v}{!l \mid \mu \rightarrow v \mid \mu} \quad \frac{\Gamma \mid \Sigma \vdash e : \text{Ref } \tau}{\Gamma \mid \Sigma \vdash !e : \tau} \text{DEREF}$	
$\frac{l \notin \text{dom}(\mu)}{\text{ref } v \mid \mu \rightarrow l \mid (\mu, l \rightarrow v)} \quad \frac{\Gamma \mid \Sigma \vdash e : \tau}{\Gamma \mid \Sigma \vdash \text{ref } e : \text{Ref } \tau} \text{REF}$	

Fig. 8. References

--

Fig. 9. Recursive Types

--

Fig. 10. Type Reconstruction

--

Fig. 11. Universals

--

Fig. 12. Existentials