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* $\tau \longrightarrow \tau$*, 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 $\tau \longrightarrow \tau$, 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

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

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

1.3 Normalization

For each type τ, we define a set R_τ of closed terms of type τ and write $R_\tau(e)$ for $e \in R_\tau$. We use N to refer to the base type `Nat`.

Definition 1.7. $R_N(e)$ if and only if e halts.

Definition 1.8. $R_{\tau_1 \to \tau_2}(e_1)$ if and only if e_1 halts and whenever $R_{\tau_1}(e_2)$ we have $R_{\tau_2}(e_1 e_2)$.

Lemma 1.9 (R_τ is invariant under evaluation). *If $e : \tau$ and $e \to e'$, then $R_\tau(e) \iff R_\tau(e')$*

Proof. QED. □

$$
\begin{array}{lll}
e & ::= \text{let } x = e \text{ in } e \mid \ldots & \text{expressions} \\
E & ::= \text{let } x = E \text{ in } e \mid \text{let } x = v \text{ in } E \mid \ldots & \text{evaluation context}
\end{array}
$$

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

Fig. 2. Let

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

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

Fig. 3. Unit

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

2.5 Variants

2.6 Fixpoints

2.7 References

We need the simply typed lambda calculus extended with unit types as in Fig 3.

$$
\begin{aligned}
v &::= (v_0, \ldots, v_n) \mid \ldots && \text{values}\\
\tau &::= (\tau_0, \ldots, \tau_n) \mid \ldots && \text{types}\\
e &::= (e_0, \ldots, e_n) \mid e.i \mid \ldots && \text{expressions}\\
E &::= (E, \ldots) \mid (v, E, \ldots) \mid \ldots && \text{evaluation context}
\end{aligned}
$$

$$
(v_0, \ldots, v_n).i \rightarrow v_i \qquad \frac{\Gamma \vdash e : (\tau_0, \ldots, \tau_n) \quad i \in \{0, \ldots, n\}}{\Gamma \vdash e.i : \tau_i}\ \textsc{Proj}
$$

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

Fig. 4. Tuples

$$
\begin{aligned}
&l && \text{labels}\\
v &::= \{l_0 = v_0, \ldots, l_n = v_n\} \mid \ldots && \text{values}\\
\tau &::= \{l_0 : \tau_0, \ldots, l_n : \tau_n\} \mid \ldots && \text{types}\\
e &::= \{l_0 = e_0, \ldots, l_n = e_n\} \mid e.l \mid \ldots && \text{expressions}\\
E &::= \{l_0 = E, \ldots, l_n = e_n\} \mid \{l_0 = v_0, \ldots, l_i = e_i, \ldots\} \mid \ldots && \text{evaluation context}
\end{aligned}
$$

$$
\{l_0 = v_0, \ldots, l_n = v_n\}.l_i \rightarrow v_i \qquad \frac{e : (l_0 : \tau_0, \ldots, l_n : \tau_n) \quad i \in \{0, \ldots, n\}}{\Gamma \vdash e.l_i : \tau_i}\ \textsc{Proj}
$$

$$
\frac{\forall i \in \{0, \ldots, n\} \quad \Gamma \vdash e_i : \tau_i}{\Gamma \vdash (l_0 = e_0, \ldots, l_n = e_n) : (l_0 : \tau_0, \ldots, l_n : \tau_n)}\ \textsc{Tuple}
$$

Fig. 5. Records

2.8 Iso-Recursive Types

3 SUBTYPING

4 SYSTEM F

4.1 Type Reconstruction

4.2 Universals

4.3 Existentials

5 ACKNOWLEDGEMENTS

Thanks to Anton Lorenzen for his guidance on the project.

$$
\begin{array}{ll}
e ::= \langle l = e \rangle \; as \; \tau \mid \text{case } e \text{ of } \langle l_i = x_i \rangle \Rightarrow e_i \mid \ldots & \text{expressions} \\
\tau ::= \langle l_0 : \tau_0, \ldots, l_n : \tau_n \rangle \mid \ldots & \text{types} \\
E ::= \text{case } \langle l_i = E \rangle \text{ as } \tau \text{ of } \langle l_i = e_i \rangle \Rightarrow e_i & \\
\quad \mid \text{case } \langle l_i = v_i \rangle \text{ as } \tau \text{ of } \langle l_i = E \rangle \Rightarrow e_i \mid \ldots & \text{evaluation contexts}
\end{array}
$$

$$
\frac{\Gamma \vdash e_j : \tau_j}{\Gamma \vdash \langle l_j = e_j \rangle \; as \; \langle l_0 : \tau_0, \ldots, l_n : \tau_n \rangle : \langle l_0 : \tau_0, \ldots, l_n : \tau_n \rangle} \; \textsc{Variant}
$$

$$
\frac{\Gamma \vdash e : \langle l_i : \tau_i \rangle \qquad \forall i \in \{0, \ldots, n\} \; \Gamma, x_i : \tau_i \vdash e_i : \tau}{\Gamma \vdash \text{case } e \text{ of } \langle l_i = x_i \rangle \Rightarrow e_i : \tau} \; \textsc{Case}
$$

$$
\text{case } \langle l_j = v_j \rangle \text{ as } \tau \text{ of } \langle l_i = x_i \rangle \Rightarrow e_i \longrightarrow e_j[v_j / x_j] \qquad i \in \{0, \ldots, n\}
$$

Fig. 6. Variants

$$
\begin{array}{ll}
e ::= \text{fix } e \mid \ldots & \text{expressions} \\
E ::= \text{fix } E \mid \ldots & \text{evaluation context}
\end{array}
$$

$$
\text{fix } \lambda x : \tau . e \longrightarrow e[(\lambda x : \tau . e) / x] \qquad \frac{\Gamma \vdash e : \tau \rightarrow \tau}{\Gamma \vdash \text{ fix } e : \tau} \; \textsc{Fix}
$$

Fig. 7. Fixpoints

l	memory locations
$e ::= \text{ref}\ e \mid\ !e \mid e := e \mid \ldots$	expressions
$v ::= l \mid \ldots$	values
$\tau ::= \texttt{Ref}\ \tau$	types
$\mu ::= \varnothing \mid \mu, l = v$	stores
$\Sigma ::= \varnothing \mid \Sigma, l : \tau$	store typing

$$\frac{\Sigma(l) = \tau}{\Gamma \mid \Sigma \vdash l : \texttt{Ref}\ \tau}\ \textsc{Loc}$$

$$l := v \mid \mu \to () \mid \mu[l \to v] \qquad \frac{\Gamma \mid \Sigma \vdash e_1 : \texttt{Ref}\ \tau \qquad \Gamma \mid \Sigma \vdash e_2 : \tau}{\Gamma \mid \Sigma \vdash e_1 := e_2 : \texttt{Unit}}\ \textsc{Assign}$$

$$\frac{\mu(l) = v}{!l \mid \mu \to v \mid \mu} \qquad \frac{\Gamma \mid \Sigma \vdash e : \texttt{Ref}\ \tau}{\Gamma \mid \Sigma \vdash\ !e : \tau}\ \textsc{Deref}$$

$$\frac{l \notin dom(\mu)}{\text{ref}\ v \mid \mu \to l \mid (\mu, l \to v)} \qquad \frac{\Gamma \mid \Sigma \vdash e : \tau}{\Gamma \mid \Sigma \vdash \text{ref}\ e : \texttt{Ref}\ \tau}\ \textsc{Ref}$$

Fig. 8. References

α	type variables
$e ::= \text{fold}\ [\tau]\ e \mid \text{unfold}\ [\tau]\ e \mid \ldots$	expressions
$v ::= \text{fold}\ [\tau]\ v \mid \ldots$	values
$\tau ::= \alpha \mid \mu\alpha.\tau \mid \ldots$	types
$E ::= \text{fold}\ [\tau]\ E \mid \text{unfold}\ [\tau]\ E \mid \ldots$	evaluation contexts

$$\frac{u = \mu\alpha.\tau_1 \quad \Gamma \vdash e : \tau_1[u/\alpha]}{\Gamma \vdash \text{fold}\ [u]\ e : u}\ \textsc{Fold} \qquad \frac{u = \mu\alpha.\tau_1 \quad \Gamma \vdash e : u}{\Gamma \vdash \text{unfold}\ [u]\ e : \tau_1[u/\alpha]}\ \textsc{Unfold}$$

$$\text{unfold}\ [\alpha_2]\ (\text{fold}\ [\alpha_1]\ v) \to v$$

Fig. 9. Iso-Recursive Types

$$\begin{array}{rll} \alpha & & \text{type variables} \\ e & ::= e[\alpha] \mid \ldots & \text{expressions} \\ v & ::= \lambda\alpha.e \mid \ldots & \text{values} \\ \tau & ::= \alpha \mid \forall\alpha.\tau \mid \ldots & \text{types} \\ \Gamma & ::= \Gamma, \alpha \mid \ldots & \text{typing contexts} \\ E & ::= E[\alpha] \mid \ldots & \text{evaluation contexts} \end{array}$$

$$(\lambda\alpha.e)[\tau] \rightarrow e[\tau/\alpha] \qquad \frac{\Gamma, \alpha \vdash e : \tau}{\Gamma \vdash \lambda\alpha.e : \forall\alpha.\tau}\ \textsc{Abs} \qquad \frac{\Gamma \vdash e : \forall\alpha.\tau_1}{\Gamma \vdash e[\tau_2] : \tau_1[\tau_2/\alpha]}\ \textsc{App}$$

Fig. 10. Universals

Fig. 11. Existentials