SECD Machine Implementation of $\lambda_{\to}$

TAZMILUR SAAD, Independent, USA

This work documents an autodidactic implementation of an interpreter for a simply typed lambda calculus extended unit, sums and product types. The interpreter is based on the SECD machine.

1 CALCULUS

Contexts are an ordered list of variables equipped with a type.

$$\Gamma \coloneqq \varnothing \mid \Gamma, x : \tau$$

The types in our system includes the unit type, sum types, product types, function types and the base types.

$$\tau \coloneqq \mathbb{1} \mid \tau + \tau \mid \tau \times \tau \mid \tau \to \tau \mid \texttt{Nat} \mid \texttt{Bool}$$

The syntax of the system includes variables, abstractions, applications, let bindings, if-then-else and members of the base types of natural numbers and booleans.

$$e \coloneqq x \mid () \mid \textsc{inl}\ e \mid \textsc{inr}\ e \mid (e_1, e_2) \mid \lambda x : \tau . e \mid e_1 e_2 \mid \mathbb{N} \mid \mathbb{B}$$
$$\mid \text{let } x = e_1 \text{ in } e_2 \mid \text{case } e \text{ of } \textsc{inl}\ x \to e_2,\ \textsc{inr}\ y \to e_2$$

The typing judgements and the operational semantics are shown in Figure 1 and 2 respectively.

2 SOUNDNESS

Lemma 2.1 (Permutation). *If $\Gamma \vdash e : \tau$ and Δ is a permutation of Γ, then $\Delta \vdash e : \tau$. Moreover, the latter derivation has the same depth as the former.*

Proof. Note that a typing context $\Gamma = (e_n : \tau_n), (e_{n-1} : \tau_{n-1}), \dots, (e_1 : \tau_1), (e_0 : \tau_0)$ is a sequence which assigns to each e_i a type τ_i. A permutation is a bijection $\Delta : \Gamma \to \Gamma$. We will use Δ as a bijection and Δ as a typing context interchangeably.

Assume $\Gamma \vdash e : \tau$ and Δ is a permutation of Γ. We proceed by structural induction on the typing derivations.

[Unit, Nat, Bool]: Let n be the length of Γ. Since Δ is a bijection, there exists some $j \leq n$ such that $\Delta(e_j : \tau_j) = e : \tau$. Therefore, $\Delta \vdash e : \tau$ and the depth does not changes.

[Var]: If $e : \tau \in \Gamma$, then $e : \tau \in \Delta$ since Δ contains the same elements as Γ. Therefore, we can apply the judgement that $\Delta \vdash e : \tau$. Moreover, the depth does not change.

[Inl, Inr, Pair, Proj 1, Proj 2, App]: In each of these cases, we can assume that the permutation property holds for the antecedent. Therefore, we can substitute $\Gamma \vdash e_i : \tau_i$ for some arbitrary i with $\Delta \vdash e_i : \tau_i$ and straightforwardly apply the judgement and notice that the depth does not change.

[Let, Case, Lam]: Each of these cases contain either Γ or $\Gamma \vdash e : \tau$ for which we can assume the permutation property. All of them extend Γ with some term $x_j : \tau_j$ but we can also extend Δ with those terms by adding a mapping from that term to itself. Therefore, Δ remains a permutation of Γ and it contains the same exact elements. Since we satisfy the assumptions, we can apply the judgements to reach the same conclusions. □

Lemma 2.2 (Weakening). *If $\Gamma \vdash e : \tau_1$ and $x \notin dom(\Gamma)$, then $\Gamma, x : \tau_2 \vdash e : \tau_1$. Moreover, the latter derivation has the same depth as the former.*

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

$$\frac{}{\Gamma \vdash () : \mathbb{1}}\ \textsc{Unit} \qquad \frac{}{\Gamma \vdash \mathbb{N} : \texttt{Nat}}\ \textsc{Nat} \qquad \frac{}{\Gamma \vdash \mathbb{B} : \texttt{Bool}}\ \textsc{Bool}$$

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

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \text{ inl } e : \tau_1 + \tau_2}\ \textsc{Inl} \qquad \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \text{ inr } e : \tau_1 + \tau_2}\ \textsc{Inr}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{ let } x = e_1 \text{ in } e_2 : \tau_2}\ \textsc{Let} \qquad \frac{\Gamma \vdash e : \tau_1 + \tau_2 \quad \Gamma, x_1 : \tau_1 \vdash y_1 : \tau_3 \quad \Gamma, x_2 : \tau_2 \vdash y_2 : \tau_3}{\Gamma \vdash \text{ case } e \text{ of } \textsc{inl}\ x_1 \rightarrow y_1 \mid \textsc{inr}\ x_2 \rightarrow y_2 : \tau_3}\ \textsc{Case}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}\ \textsc{Pair} \qquad \frac{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}{\Gamma \vdash e_1 : \tau_1}\ \textsc{Proj 1} \qquad \frac{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}{\Gamma \vdash e_2 : \tau_2}\ \textsc{Proj 2}$$

Fig. 1. Simply typed lambda calculus

$$\frac{e_1 \rightarrow e'_1}{e_1 e_2 \rightarrow e'_1 e_2}\ \textsc{App 1} \qquad \frac{e_2 \rightarrow e'_2}{v_1 e_2 \rightarrow v e'_2}\ \textsc{App 2} \qquad \frac{e \rightarrow e'}{\text{let } x = e \text{ in } e_2 \rightarrow \text{let } x = e' \text{ in } e_2}\ \textsc{Let}$$

$$\frac{e \rightarrow e'}{e.1 \rightarrow e'.1}\ \textsc{Proj 1} \qquad \frac{e \rightarrow e'}{e.2 \rightarrow e'.2}\ \textsc{Proj 2} \qquad \frac{e_1 \rightarrow e'_1}{(e_1, e_2) \rightarrow (e'_1, e_2)}\ \textsc{Pair 1}$$

$$\frac{e_2 \rightarrow e'_2}{(v_1, e_2) \rightarrow (v_1, e'_2)}\ \textsc{Pair 2} \qquad \frac{e \rightarrow e'}{\textsc{inl}\ e \rightarrow \ \textsc{inl}\ e'}\ \textsc{Inl} \qquad \frac{e \rightarrow e'}{\textsc{inr}\ e \rightarrow \ \textsc{inr}\ e'}\ \textsc{Inr}$$

$$\begin{array}{rr}
(\lambda x : \tau . e) v \rightarrow [x \mapsto v] e & \textsc{AppAbs} \\
(v_1, v_2).1 \rightarrow v_1 & \textsc{PairBeta1} \\
(v_1, v_2).2 \rightarrow v_2 & \textsc{PairBeta2} \\
\text{case } (\ \textsc{inl}\ v\ ) \text{ of } \textsc{inl}\ x_1 \Rightarrow e_1 \mid \textsc{inr}\ x_2 \Rightarrow t_2 \rightarrow [x_1 \mapsto v] e_1 & \textsc{CaseInl} \\
\text{case } (\ \textsc{inr}\ v\ ) \text{ of } \textsc{inl}\ x_1 \Rightarrow e_1 \mid \textsc{inr}\ x_2 \Rightarrow t_2 \rightarrow [x_2 \mapsto v] e_1 & \textsc{CaseInr}
\end{array}$$

Fig. 2. Small Step Operational Semantics

Proof. We assume that extension of a context does not result in naming conflicts. We proceed by structural induction on the typing derivation.

[Unit, Nat, Bool]: These judgements do not assume anything about the context. If we extend the context with a non-existing element, we can reach the same conclusions. Moreover, there are no subterms so the derivation tree's depth does not change.

[Var]: If we extend Γ with a non-existing element, $x : \tau \in \Gamma$ still holds and we can apply the judgement.

[Inl, Inr, Pair, Proj 1, Proj 2, App]: We assume that the weakening lemma holds for the antecedent These rules do not extend Γ, so addition of another element allows us to reach the same conclusion.

[Let, Case, Lam]: We assume the weakening lemma holds for the antecedent. Since these rules extend Γ, there are two cases. If we extend Γ with a variable that has the same type as the assumptions of the judgement, then we get that inference for free. If we extend Γ with a different type, then the variables in the original assumption will still exist modulo renaming and we can still apply the judgement. □

Theorem 2.3 (Progress). *If $\cdot \vdash x : \tau$ is a well typed term then x is a value or there exists some y such that $x \mapsto y$.*

Proof. Admitted. □

Theorem 2.4 (Preservation). *If $\cdot \vdash x : \tau$ and $x \mapsto y$, then $\cdot \vdash y : \tau$.*

Proof. Admitted. □

3 ACKNOWLEDGEMENTS

Thanks to Anton Lorenzen for his guidance on the project.