Lecture #17: Typed $\lambda$-calculi

CS 6371: Advanced Programming Languages
March 31, 2022

There are many typed $\lambda$-calculi, of which we will study two. The first is known as the simply-typed $\lambda$-calculus or $\lambda_\to$. Here are some of the operations commonly included in $\lambda_\to$.

\[
e ::= n \quad \text{integers} \\
    | v \quad \text{variables} \\
    | \lambda v: \tau \cdot e \quad \text{abstraction} \\
    | e_1 e_2 \quad \text{application} \\
    | \text{true} | \text{false} \quad \text{booleans} \\
    | e_1 \ aop \ e_2 \quad \text{arithmetic ops} \\
    | e_1 \ bop \ e_2 \quad \text{boolean ops} \\
    | e_1 \ cmp \ e_2 \quad \text{integer comparison} \\
    | (e_1, e_2) \quad \text{pairs} \\
    | \pi_1 e \ | \pi_2 e \quad \text{projection} \\
    | () \quad \text{unit} \\
    | \text{in}_{\tau_1} e \ | \text{in}_{\tau_2} e \quad \text{injection} \\
    | (\text{case } e \text{ of } \text{in}_{\tau_1}(v_1) \to e_1 \ | \text{in}_{\tau_2}(v_2) \to e_2) \quad \text{case distinction}
\]

The type system for the above language is as follows.

\[
\tau ::= \text{int} \quad \text{integer} \\
    | \text{bool} \quad \text{boolean} \\
    | \tau_1 \to \tau_2 \quad \text{function} \\
    | \tau_1 \times \tau_2 \quad \text{product type} \\
    | \text{unit} \quad \text{unit type} \\
    | \tau_1 + \tau_2 \quad \text{sum type} \\
    | \text{void} \quad \text{uninhabited type}
\]
Since simply-typed $\lambda$-calculus is strongly normalizing, it is sometimes augmented with an explicit fixed-point operator fix:

$$e ::= \cdots \mid \text{fix } e$$

$$e \rightarrow e'$$

$$\text{fix } e \rightarrow \text{fix } e'$$

$$\text{fix}(\lambda v : \tau.e) \rightarrow_{\text{f}} e[\text{fix}(\lambda v : \tau.e)/v]$$

$$\Gamma \vdash e : (\tau \rightarrow \tau') \rightarrow (\tau \rightarrow \tau')$$

$$\Gamma \vdash \text{fix } e : \tau \rightarrow \tau'$$

System $F$ [Girard 1972, Reynolds 1974], also known as the polymorphic $\lambda$-calculus, extends the simply-typed $\lambda$-calculus with type variables. Like regular variables, the type variables must not appear unbound (free).

$$e ::= \cdots \mid \Lambda \alpha.e$$

poly morphic abstraction

$$| e[\tau]$$

poly morphic instantiation

$$\tau ::= \cdots \mid \alpha$$

type variables

$$| \forall \alpha.\tau$$

universal types

$$(\Lambda \alpha.e)[\tau] \rightarrow_{\text{f}} e[\tau/\alpha]$$

$$\Gamma \vdash e : \tau$$

(16) $$\Gamma \vdash \Lambda \alpha.e : \forall \alpha.\tau$$

$$\Gamma \vdash e : \forall \alpha.\tau'$$

(17) $$\Gamma \vdash e[\tau] : \tau'[\tau/\alpha]$$