Simply-typed Lambda Calculus

Stefan Muller

CS 5095: Types and Programming Languages, Fall 2025
Lectures 9-10

1 Function and Unit Types

We're going to start adding types to the (formerly untyped) lambda calculus. The first type we'll add is a type of functions $\tau_1 \to \tau_2$, which is a function from arguments of type τ_1 to results of type τ_2. The arrow will associate to the right, so the type $\tau_1 \to \tau_2 \to \tau_3$ means a function that takes a τ_1 and returns a function that takes a τ_2 and returns a τ_3, which is how we implemented multi-argument functions in lambda calculus. The types τ_1 and τ_2 can themselves be functions, but this recursion needs to bottom out somewhere, so we need another type. We'll add a "base type" unit (sometimes notated 1) with one value () (also pronounced "unit"). In C, you might think of this as the type "void." We'll see soon why we **don't** call this "void." Expressions e are the same as lambda terms but now we actually say what the type of a function argument is (like we do in C, Java, etc.)

We'll keep adding to this, but this is the language for now:

$$\begin{array}{lll} \textit{Expressions} & e & ::= & x \mid () \mid \lambda x : \tau.e \mid e\ e \\ \textit{Types} & \tau & ::= & \mathsf{unit} \mid \tau \to \tau \end{array}$$

And here are the static and dynamic rules for now:

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ (\textsc{Var}) \qquad \frac{}{\Gamma \vdash () : \mathsf{unit}}\ (\mathsf{unit}\text{-I}) \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}\ (\to\text{-E})$$

$$\frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda x : \tau.e : \tau \to \tau'}\ (\to\text{-I}) \qquad \frac{}{()\ \mathsf{val}}\ (\textsc{ValUnit}) \qquad \frac{}{\lambda x : \tau.e\ \mathsf{val}}\ (\textsc{ValLambda})$$

$$\frac{e_1 \mapsto e_1'}{e_1\ e_2 \mapsto e_1'\ e_2}\ (\textsc{StepSearchAppLeft}) \qquad \frac{e_2 \mapsto e_2'}{(\lambda x : \tau.e)\ e_2 \mapsto (\lambda x : \tau.e)\ e_2'}\ (\textsc{StepSearchAppRight})$$

$$\frac{v\ \mathsf{val}}{(\lambda x : \tau.e)\ v \mapsto [v/x]e}\ (\textsc{StepApp})$$

Note that for the statics, instead of using more descriptive names, we're now labeling the rules with the type they deal with ($\to$ or unit) and whether they Introduce or Eliminate expressions of that type.

Example.

$$(\lambda x : \mathsf{unit}.x)\ ()$$

$$\frac{\dfrac{\dfrac{}{x : \mathsf{unit} \vdash x : \mathsf{unit}}\ (\textsc{Var})}{\bullet \vdash \lambda x : \mathsf{unit}.x : \mathsf{unit} \to \mathsf{unit}}\ (\to\text{-I}) \qquad \dfrac{}{\bullet \vdash () : \mathsf{unit}}\ (\mathsf{unit}\text{-I})}{\bullet \vdash (\lambda x : \mathsf{unit}.x)\ () : \mathsf{unit}}\ (\to\text{-E})$$

$$\begin{array}{rl} & (\lambda x : \mathsf{unit}.x)\ () \\ \mapsto & () \end{array}$$

Example.

$$(\lambda f : \mathsf{unit} \to \mathsf{unit}.f\ ())\ (\lambda x : \mathsf{unit}.x) : \mathsf{unit}$$

$(\lambda f : \mathsf{unit} \to \mathsf{unit}.f\ ())\ (\lambda x : \tau.x)$ for any $\tau \neq \mathsf{unit}$ isn't well-typed: the type annotations matter!

2 Products

$$\begin{array}{llll} \textit{Expressions} & e & ::= & \cdots \mid (e, e) \mid \mathsf{fst}\ e \mid \mathsf{snd}\ e \\ \textit{Types} & \tau & ::= & \cdots \mid \tau \times \tau \end{array}$$

Examples (Let's assume we also have int and string as base types for these.)

$$\begin{array}{lcl} (\overline{1}, \overline{2}) & : & \mathsf{int} \times \mathsf{int} \\ (\overline{1}, \text{“Hi”}) & : & \mathsf{int} \times \mathsf{string} \\ (\text{“Hi”}, \overline{1}) & : & \mathsf{string} \times \mathsf{int} \\ ((), \overline{1}) & : & \mathsf{unit} \times \mathsf{int} \\ \lambda x : \mathsf{unit}.\lambda y : \mathsf{int}.(x, y) & : & \mathsf{unit} \to \mathsf{int} \to (\mathsf{unit} \times \mathsf{int}) \\ ((\lambda x : \mathsf{unit}.x), \overline{1}) & : & ((\mathsf{unit} \to \mathsf{unit}), \mathsf{int}) \\ \lambda x : \mathsf{int} \times \mathsf{string}.\mathsf{fst}\ x & : & \mathsf{int} \end{array}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}\ (\times\text{-I}) \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{fst}\ e : \tau_1}\ (\times\text{-E1}) \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{snd}\ e : \tau_2}\ (\times\text{-E2})$$

$$\frac{v_1\ \mathsf{val} \qquad v_2\ \mathsf{val}}{(v_1, v_2)\ \mathsf{val}}\ (\textsc{ValPair}) \qquad \frac{e_1 \mapsto e_1'}{(e_1, e_2) \mapsto (e_1', e_2)}\ (\textsc{StepSearchPairLeft})$$

$$\frac{v_1\ \mathsf{val} \qquad e_2 \mapsto e_2'}{(v_1, e_2) \mapsto (v_1, e_2')}\ (\textsc{StepSearchPairRight}) \qquad \frac{e \mapsto e'}{\mathsf{fst}\ e \mapsto \mathsf{fst}\ e'}\ (\textsc{StepSearchFst})$$

$$\frac{v_1\ \mathsf{val} \qquad v_2\ \mathsf{val}}{\mathsf{fst}\ (v_1, v_2) \mapsto v_1}\ (\textsc{StepFst}) \qquad \frac{e \mapsto e'}{\mathsf{snd}\ e \mapsto \mathsf{snd}\ e'}\ (\textsc{StepSearchSnd}) \qquad \frac{v_1\ \mathsf{val} \qquad v_2\ \mathsf{val}}{\mathsf{snd}\ (v_1, v_2) \mapsto v_2}\ (\textsc{StepSnd})$$

3 Sums

$$\begin{array}{llll} \textit{Expressions} & e & ::= & \cdots \mid \mathsf{inl}\ e \mid \mathsf{inr}\ e \mid \mathsf{case}\ e\ \mathsf{of}\ \{x.e; y.e\} \\ \textit{Types} & \tau & ::= & \tau + \tau \end{array}$$

$\mathsf{int} + \mathsf{string}$: can be an int or a string (but you have to say which one)

Examples.

$$\begin{array}{lcl} \mathsf{inl}\ \overline{1} & : & \mathsf{int} + \mathsf{string} \\ \mathsf{inr}\ \text{“Hi”} & : & \mathsf{int} + \mathsf{string} \\ \mathsf{case}\ \mathsf{inl}\ \overline{1}\ \mathsf{of}\ \{x.x; y.|y|\} & : & \mathsf{int} \\ \mathsf{case}\ \mathsf{inr}\ \text{“Hi”}\ \mathsf{of}\ \{x.x; y.|y|\} & : & \mathsf{int} \\ \lambda x : \mathsf{int} + \mathsf{string}.\mathsf{case}\ x\ \mathsf{of}\ \{x.x; y.|y|\} & : & (\mathsf{int} + \mathsf{string}) \to \mathsf{int} \end{array}$$

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathsf{inl}\ e : \tau_1 + \tau_2}\ (\text{+-I1}) \qquad \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathsf{inr}\ e : \tau_1 + \tau_2}\ (\text{+-I2})$$

$$\frac{\Gamma \vdash e : \tau_1 + \tau_2 \qquad \Gamma, x : \tau_1 \vdash e_1 : \tau \qquad \Gamma, y : \tau_2 \vdash e_2 : \tau}{\Gamma \vdash \mathsf{case}\ e\ \mathsf{of}\ \{x.e_1; y.e_2\} : \tau}\ (\text{+-E}) \qquad \frac{v\ \mathsf{val}}{\mathsf{inl}\ v\ \mathsf{val}}\ (\textsc{ValInL})$$

$$\frac{v\ \mathsf{val}}{\mathsf{inr}\ v\ \mathsf{val}}\ (\textsc{ValInR}) \qquad \frac{e \mapsto e'}{\mathsf{inl}\ e \mapsto \mathsf{inl}\ e'}\ (\textsc{StepSearchInL}) \qquad \frac{e \mapsto e'}{\mathsf{inr}\ e \mapsto \mathsf{inr}\ e'}\ (\textsc{StepSearchInR})$$

$$\frac{e \mapsto e'}{\mathsf{case}\ e\ \mathsf{of}\ \{x.e_1; y.e_2\} \mapsto \mathsf{case}\ e'\ \mathsf{of}\ \{x.e_1; y.e_2\}}\ (\textsc{StepSearchCase})$$

$$\frac{v\ \mathsf{val}}{\mathsf{case}\ \mathsf{inl}\ v\ \mathsf{of}\ \{x.e_1; y.e_2\} \mapsto [v/x]e_1}\ (\textsc{StepCaseL}) \qquad \frac{v\ \mathsf{val}}{\mathsf{case}\ \mathsf{inr}\ v\ \mathsf{of}\ \{x.e_1; y.e_2\} \mapsto [v/y]e_2}\ (\textsc{StepCaseR})$$

$$\begin{array}{ll} & \mathsf{case}\ \mathsf{inr}\ \text{“Hi”}\ \mathsf{of}\ \{x.x; y.|y|\} \\ \mapsto & |\text{“Hi”}| \\ \mapsto & \overline{2} \end{array}$$

Example: Options (like null pointer)

$$\mathsf{int\ option} \triangleq \mathsf{int} + ()$$

$$\mathsf{case}\ e\ \mathsf{of}\ \{x.x; y.\overline{0}\}$$

4 Type Safety

Lemma 1 (Canonical Forms). *1. If e val and $\bullet \vdash e : \tau_1 \times \tau_2$, then $e = (v_1, v_2)$ where v_1 val and v_2 val.*

2. If e val and $\bullet \vdash e : \tau_1 + \tau_2$, then $e = \mathsf{inl}\ v$ where v val or $e = \mathsf{inr}\ v$ where v val.

Theorem 1 (Progress). *If $\bullet \vdash e : \tau$ then either e val or there exists e' such that $e \mapsto e'$.*

Proof. By induction on the derivation of $\bullet \vdash e : \tau$. We'll just do one case as an example.

- +-E. Then $e = \mathsf{case}\ e_1\ \mathsf{of}\ \{x.e_2; y.e_3\}$ and $\bullet \vdash e_1 : \tau_1 + \tau_2$ and $x : \tau_1 \vdash e_2 : \tau$ and $y : \tau_2 \vdash e_3 : \tau$. By induction, e_1 val or there exists $e_1 \mapsto e_1'$.
  - e_1 val. Then, by canonical forms, $e_1 = \mathsf{inl}\ v$ or $e_2 = \mathsf{inr}\ v$, and e steps by (StepCaseL) or (StepCaseR).
  - $e_1 \mapsto e_1'$. Then e steps by (StepSearchCase).

□

Theorem 2 (Preservation). *If $\bullet \vdash e : \tau$ and $e \mapsto e'$ then $\bullet \vdash e' : \tau$.*

Proof. By induction on the derivation of $e \mapsto e'$. We'll just do a few cases.

- StepSearchAppLeft. Then $e = e_1\ e_2$. By inversion, $\bullet \vdash e_1 : \tau' \to \tau$. By induction, $\bullet \vdash e_1' : \tau' \to \tau$. Apply $\to$-E.
- StepSearchAppRight. Similar.
- StepApp. Then $e = (\lambda x : \tau'.e_0)\ v$ and $e' = [v/x]e_0$. By inversion on $\to$-E, $\bullet \vdash \lambda x : \tau'.e_0 : \tau' \to \tau$ and $\bullet \vdash v : \tau'$. By inversion on $\to$-I, $x : \tau' \vdash e_0 : \tau$. By substitution, $\bullet \vdash [v/x]e_0 : \tau$.

- StepFst. Then $e = \mathsf{fst}\ (e', v_2)$. By inversion on $\times$-E1, $\bullet \vdash (e', v_2) : \tau \times \tau_2$. By inversion on $\times$-I, $\bullet \vdash e' : \tau$.

- StepCaseL. Then $e = \mathsf{case}\ \mathsf{inl}\ v\ \mathsf{of}\ \{x.e_1; y.e_2\}$ and $v\ \mathsf{val}$ and $e' = [v/x]e_1$. By inversion, $\bullet \vdash \mathsf{inl}\ v : \tau_1 + \tau_2$ and $x : \tau_1 \vdash e_1 : \tau$. By another inversion, $\bullet \vdash v : \tau_1$. By substitution, $\bullet \vdash [v/x]e_1 : \tau$.

□

Fun fact If $\bullet \vdash e : \tau$ then there exists v such that $v\ \mathsf{val}$ and $e \mapsto^* v$. (We won't prove it though, that proof is actually pretty tricky and uses some techniques we probably won't see in the class.)

Remember that this wasn't true for the untyped lambda calculus. So why is it true now that we've added types? Remember our self-application trick. Let's try to figure out the type of $(\lambda x.x\ x)\ (\lambda x.x\ x)$. There's no type annotation, so let's just say the type of x is τ and we'll figure it out later. We'll also say the return type of each function is τ'.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x:\tau \vdash x:\tau\to\tau}\,(???) \qquad x:\tau\vdash x:\tau}{x:\tau\vdash x\ x:\tau'}}{\bullet\vdash \lambda x.x\ x:\tau\to\tau'} \qquad \dots}{\bullet \vdash (\lambda x.x\ x)\ (\lambda x.x\ x) :???}$$

5 Observations

Booleans

$$\begin{array}{lcl}
\mathsf{bool} & \triangleq & \mathsf{unit} + \mathsf{unit} \\
\mathsf{true} & \triangleq & \mathsf{inl}\ () \\
\mathsf{false} & \triangleq & \mathsf{inr}\ () \\
\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3\ \mathsf{fi} & \triangleq & \mathsf{case}\ e_1\ \mathsf{of}\ \{x.e_2; y.e_3\}
\end{array}$$

- The type of Booleans is sometimes called 2 because it has 2 elements.
- We write $\cong$ to mean that two types are "equal" (really isomorphic, but we're not going to go into the definition of that).
- So $2 \cong 1 + 1$!
- This isn't a coincidence. Think about how many elements, e.g., the type 2×2 has (i.e., how many different pairs of Booleans there are).

More general sums and products

- *Binary products* $\tau_1 \times \tau_2$:
  - 1 intro form (way to create)—pair
  - 2 elim forms (things to do with them)—projections fst, snd
- n-ary products $\tau_1 \times \cdots \times \tau_n$:
  - 1 intro form (way to create)—"tuple"
  - n elim forms (things to do with them)—projections π_i
  - (We can also just encode this as nested binary products: $\tau_1 \times (\tau_2 \times (\cdots \times (\tau_{n-1} \times \tau_n))))$
- *Binary sums* $\tau_1 + \tau_2$:
  - 2 intro forms (inl, inr)

  - 1 elim form (case)

- n-ary sums $\tau_1 + \cdots + \tau_n$:
  - n intro forms ("injections")
  - 1 elim form (case)
- 0-ary ("nullary") product?
  - 1 intro form (way to create)—"tuple"
  - 0 elim forms—nothing to do with it
  - Unit!
- Nullary sum?
  - 0 intro forms
  - 1 thing to do
  - void

6 Inference Rules (including void)

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ (\textsc{Var}) \qquad \frac{}{\Gamma \vdash () : \mathsf{unit}}\ (\mathsf{unit}\text{-I}) \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}\ (\to\text{-E})$$

$$\frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda x : \tau.e : \tau \to \tau'}\ (\to\text{-I}) \qquad \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}\ (\times\text{-I}) \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{fst}\ e : \tau_1}\ (\times\text{-E1})$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{snd}\ e : \tau_2}\ (\times\text{-E2}) \qquad \frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathsf{inl}\ e : \tau_1 + \tau_2}\ (+\text{-I1}) \qquad \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathsf{inr}\ e : \tau_1 + \tau_2}\ (+\text{-I2})$$

$$\frac{\Gamma \vdash e : \tau_1 + \tau_2 \qquad \Gamma, x : \tau_1 \vdash e_1 : \tau \qquad \Gamma, y : \tau_2 \vdash e_2 : \tau}{\Gamma \vdash \mathsf{case}\ e\ \mathsf{of}\ \{x.e_1; y.e_2\} : \tau}\ (+\text{-E}) \qquad \frac{\Gamma \vdash e : \mathsf{void}}{\Gamma \vdash \mathsf{abort}\ e : \tau}\ (\mathsf{void}\text{-E})$$

$$\frac{}{()\ \mathsf{val}}\ (\textsc{ValUnit}) \qquad \frac{}{\lambda x : \tau.e\ \mathsf{val}}\ (\textsc{ValLambda}) \qquad \frac{v_1\ \mathsf{val} \qquad v_2\ \mathsf{val}}{(v_1, v_2)\ \mathsf{val}}\ (\textsc{ValPair})$$

$$\frac{v\ \mathsf{val}}{\mathsf{inl}\ v\ \mathsf{val}}\ (\textsc{ValInL}) \qquad \frac{v\ \mathsf{val}}{\mathsf{inr}\ v\ \mathsf{val}}\ (\textsc{ValInR})$$

$$\frac{e_1 \mapsto e_1'}{e_1\ e_2 \mapsto e_1'\ e_2}\ (\textsc{StepSearchAppLeft}) \qquad \frac{e_2 \mapsto e_2'}{(\lambda x : \tau.e)\ e_2 \mapsto (\lambda x : \tau.e)\ e_2'}\ (\textsc{StepSearchAppRight})$$

$$\frac{v\ \mathsf{val}}{(\lambda x : \tau.e)\ v \mapsto [v/x]e}\ (\textsc{StepApp}) \qquad \frac{e_1 \mapsto e_1'}{(e_1, e_2) \mapsto (e_1', e_2)}\ (\textsc{StepSearchPairLeft})$$

$$\frac{v_1\ \mathsf{val} \qquad e_2 \mapsto e_2'}{(v_1, e_2) \mapsto (v_1, e_2')}\ (\textsc{StepSearchPairRight}) \qquad \frac{e \mapsto e'}{\mathsf{fst}\ e \mapsto \mathsf{fst}\ e'}\ (\textsc{StepSearchFst})$$

$$\frac{v_1\ \mathsf{val} \qquad v_2\ \mathsf{val}}{\mathsf{fst}\ (v_1, v_2) \mapsto v_1}\ (\textsc{StepFst}) \qquad \frac{e \mapsto e'}{\mathsf{snd}\ e \mapsto \mathsf{snd}\ e'}\ (\textsc{StepSearchSnd}) \qquad \frac{v_1\ \mathsf{val} \qquad v_2\ \mathsf{val}}{\mathsf{snd}\ (v_1, v_2) \mapsto v_2}\ (\textsc{StepSnd})$$

$$\frac{e \mapsto e'}{\mathsf{inl}\ e \mapsto \mathsf{inl}\ e'}\ (\textsc{StepSearchInL}) \qquad \frac{e \mapsto e'}{\mathsf{inr}\ e \mapsto \mathsf{inr}\ e'}\ (\textsc{StepSearchInR})$$

$$\frac{e \mapsto e'}{\mathsf{case}\ e\ \mathsf{of}\ \{x.e_1; y.e_2\} \mapsto \mathsf{case}\ e'\ \mathsf{of}\ \{x.e_1; y.e_2\}}\ (\textsc{StepSearchCase})$$

$$\frac{v\ \mathsf{val}}{\mathsf{case}\ \mathsf{inl}\ v\ \mathsf{of}\ \{x.e_1; y.e_2\} \mapsto [v/x]e_1}\ (\textsc{StepCaseL}) \qquad \frac{v\ \mathsf{val}}{\mathsf{case}\ \mathsf{inr}\ v\ \mathsf{of}\ \{x.e_1; y.e_2\} \mapsto [v/y]e_2}\ (\textsc{StepCaseR})$$

$$\frac{e \mapsto e'}{\mathsf{abort}\ e \mapsto \mathsf{abort}\ e'}\ (\textsc{StepSearchAbort})$$