

Extensions to the Simply Typed Lambda Calculus

- ▶ unit and sequencing
- ▶ let binding
- ▶ pairs
- ▶ records
- ▶ variants
- ▶ recursive functions

Unit and Sequencing

$$\begin{aligned} e & ::= \dots \mid \mathit{unit} \mid e; e \\ T & ::= \dots \mid \mathit{Unit} \end{aligned}$$

Typing rule for unit:

$$\frac{}{\Gamma \vdash \mathit{unit} : \mathit{Unit}}$$

Definition of sequencing $(e_1; e_2)$ by translation:

$$e_1; e_2 \equiv ((\lambda x : \mathit{Unit}. e_2) e_1)$$

where x is a fresh variable (not in $FV(e_2)$).

Let Binding (local variables)

$$e ::= \dots \mid \text{let } x = e_1 \text{ in } e_2$$

For evaluation purposes, we can use:

$$\text{let } x = e_1 \text{ in } e_2 \equiv ((\lambda x. e_2) e_1)$$

However, notice that the parameter x wasn't given a type. We can give it a type, but we don't know that type until we've type checked e_1 . So we can't just expand a let into an application and λ before we do type checking.

$$\frac{\Gamma \vdash e_1 : T_1 \quad \Gamma, x : T_1 \vdash e_2 : T_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : T_2}$$

$$e ::= \dots \mid (e, e) \mid \text{fst } e \mid \text{snd } e$$

$$T ::= \dots \mid T \times T$$

Reduction rules:

$$\text{fst } (v_1, v_2) \longrightarrow v_1$$

$$\text{snd } (v_1, v_2) \longrightarrow v_2$$

Evaluation contexts:

$$E ::= \dots \mid (E, e) \mid (v, E)$$

Typing rules:

$$\frac{\Gamma \vdash e_1 : T_1 \quad \Gamma \vdash e_2 : T_2}{\Gamma \vdash (e_1, e_2) : T_1 \times T_2}$$

$$\frac{\Gamma \vdash e : T_1 \times T_2}{\Gamma \vdash \text{fst } e : T_1} \qquad \frac{\Gamma \vdash e : T_1 \times T_2}{\Gamma \vdash \text{snd } e : T_2}$$

$$\begin{aligned} e & ::= \dots \mid \{l_1 = e_1, \dots, l_n = e_n\} \mid e.l \\ T & ::= \dots \mid \{l_1 : T_1, \dots, l_n : T_n\} \end{aligned}$$

Reduction rules:

$$\{l_1 = v_1, \dots, l_n = v_n\}.l_i \longrightarrow v_i \quad (1 \leq i \leq n)$$

Evaluation contexts:

$$E ::= \dots \mid \{l_1 = v_1, \dots, l_i = E, l_{i+1} = e_{i+1}, \dots, l_n = e_n\}$$

Typing rules:

$$\frac{\Gamma \vdash e_1 : T_1 \quad \dots \quad \Gamma \vdash e_n : T_n}{\Gamma \vdash \{l_1 = e_1, \dots, l_n = e_n\} : \{l_1 : T_1, \dots, l_n : T_n\}}$$
$$\frac{\Gamma \vdash e : \{l_1 : T_1, \dots, l_n : T_n\} \quad 1 \leq i \leq n}{\Gamma \vdash e.l_i : T_i}$$

Variants

$$e ::= \dots \mid T \langle l = e \rangle \mid \text{case } e \text{ of } (l_1 = x_1 \Rightarrow e_1) \cdots (l_n = x_n \Rightarrow e_n)$$
$$T ::= \dots \mid \langle l_1 : T_1, \dots, l_n : T_n \rangle$$

Reduction rules:

$$\text{case } (T \langle l_i = v \rangle) \text{ of } (l_1 = x_1 \Rightarrow e_1) \cdots (l_n = x_n \Rightarrow e_n) \longrightarrow [x_i \mapsto v]e_i$$
$$(1 \leq i \leq n)$$

Typing rules:

$$\frac{\Gamma \vdash e : T_i \quad 1 \leq i \leq n \quad T = \langle l_1 : T_1, \dots, l_n : T_n \rangle}{\Gamma \vdash T \langle l_i = e \rangle : T}$$
$$\frac{\Gamma \vdash e : \langle l_1 : T_1, \dots, l_n : T_n \rangle \quad \forall i \in \{1 \dots n\}. \Gamma, x_i : T_i \vdash e_i : T}{\Gamma \vdash \text{case } e \text{ of } (l_1 = x_1 \Rightarrow e_1) \cdots (l_n = x_n \Rightarrow e_n) : T}$$

Recursive Functions

$e ::= \dots \mid \text{fix } e \mid \text{letrec } x : T = e \text{ in } e$

Reduction rules:

$\text{fix } (\lambda f : T. e) \longrightarrow [f \mapsto (\text{fix}(\lambda f : T. e))]e$

Evaluation contexts:

$E ::= \dots \mid \text{fix } E$

Typing rules:

$$\frac{\Gamma \vdash e : T \rightarrow T}{\Gamma \vdash \text{fix } e : T}$$

Definition of letrec by translation:

$\text{letrec } x : T = e_1 \text{ in } e_2 \equiv \text{let } x = \text{fix}(\lambda x : T. e_1) \text{ in } e_2$

Example: the sum function

<pre>letrec sum : nat → nat = (λ n : nat. if (iszero n) then 0 else n + sum (pred n)) in sum 2</pre>	<pre>let sum = fix(λ sum: nat → nat. (λ n : nat. if (iszero n) then 0 else n + sum (pred n))) in sum 2</pre>
---	--

Example: running the sum function

$B \equiv (\lambda n : \text{nat. if (iszero } n) \text{ then } 0 \text{ else } n + \text{sum (pred } n))$
 $S \equiv \text{fix } (\lambda \text{ sum: nat} \rightarrow \text{nat. } B)$

sum 2

\mapsto

S 2

\mapsto

[sum \mapsto S] B 2

=

$(\lambda n : \text{nat. if (iszero } n) \text{ then } 0 \text{ else } n + S (\text{pred } n))$ 2

\mapsto

if (iszero 2) then 0 else 2 + S (pred 2)

\mapsto

2 + S (pred 2)

\mapsto

2 + [sum \mapsto S] B (pred 2)

Example: typing the sum function

$B \equiv (\lambda n : \text{nat}. \text{if } (\text{iszero } n) \text{ then } 0 \text{ else } n + \text{sum}(\text{pred } n))$

$$\frac{\frac{\vdots}{\Gamma, \text{sum} : \text{nat} \rightarrow \text{nat} \vdash B : \text{nat} \rightarrow \text{nat}}}{\Gamma \vdash (\lambda \text{sum} : \text{nat} \rightarrow \text{nat}. B) : (\text{nat} \rightarrow \text{nat}) \rightarrow (\text{nat} \rightarrow \text{nat})}}{\Gamma \vdash \text{fix } (\lambda \text{sum} : \text{nat} \rightarrow \text{nat}. B) : \text{nat} \rightarrow \text{nat}}$$