Lecture 16 – Type Extensions

AAA551: Programming Language Theory

Jihyeok Park

2026 Spring

Recall

- Simply-Typed Lambda Calculus
  - Abstract Syntax
  - Dynamic Semantics
  - Static Semantics
- Type Soundness
  - Progress
  - Preservation

Contents

Contents

Product Types

Let's extend our language with **product types** (or **pair types**).

$$\begin{array}{ll} \text{Expressions} & e ::= \cdots \mid (e_1, e_2) \mid e.1 \mid e.2 \\ \text{Values} & v ::= \ldots \mid (v_1, v_2) \end{array}$$

$$\boxed{e \to e}$$

$$\frac{}{(v_1, v_2).1 \to v_1} \qquad \frac{}{(v_1, v_2).2 \to v_2}$$

$$E ::= \ldots \mid (E, e) \mid (v, E) \mid E.1 \mid E.2$$

Product Types

PLRG

Types $\tau ::= \cdots \mid \tau_1 \times \tau_2$

$$\boxed{\Gamma \vdash e : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash e.1 : \tau_1} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash e.2 : \tau_2}$$

Sum Types

Let's extend our language with **sum types** (or **tagged union types**).

$$\begin{array}{lll} \text{Expressions} & e ::= \cdots \mid \texttt{inl}\ e \mid \texttt{inr}\ e \\ & \quad \mid \texttt{case}\ e\ \texttt{of inl}\ x \texttt{ => } e_1 \texttt{ | inr}\ y \texttt{ => } e_2 \\ \text{Values} & v ::= \ldots \mid \texttt{inl}\ v \mid \texttt{inr}\ v \end{array}$$

$$\boxed{e \to e}$$

$$\frac{}{(\texttt{case}\ (\texttt{inl}\ v)\ \texttt{of inl}\ x \texttt{ => } e_1 \texttt{ | inr}\ y \texttt{ => } e_2) \to e_1[x \mapsto v]}$$

$$\frac{}{(\texttt{case}\ (\texttt{inr}\ v)\ \texttt{of inl}\ x \texttt{ => } e_1 \texttt{ | inr}\ y \texttt{ => } e_2) \to e_2[y \mapsto v]}$$

$$E ::= \ldots \mid \texttt{inl}\ E \mid \texttt{inr}\ E \mid \texttt{case}\ E\ \texttt{of inl}\ x \texttt{ => } e_1 \texttt{ | inr}\ y \texttt{ => } e_2$$

Sum Types

PLRG

Types $\tau ::= \cdots \mid \tau_1 + \tau_2$

$$\boxed{\Gamma \vdash e : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau_1}{\Gamma \vdash \texttt{inl}\ e_1 : \tau_1 + \tau_2} \qquad \frac{\Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{inr}\ e_2 : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_1 + \tau_2 \qquad \Gamma[x : \tau_1] \vdash e_1 : \tau \qquad \Gamma[y : \tau_2] \vdash e_2 : \tau}{\Gamma \vdash (\texttt{case}\ e\ \texttt{of inl}\ x\ \texttt{=>}\ e_1\ \texttt{|}\ \texttt{inr}\ y\ \texttt{=>}\ e_2) : \tau}$$

Algebraic Data Types

Let's extend our language with **algebraic data types** (ADTs).

$$
\begin{array}{lll}
\text{Expressions} & e ::= \cdots \mid \langle x \rangle & \\
& \quad \mid \ \texttt{enum}\ t\ \{\ [\texttt{case}\ x(\tau^*)]^+\ \};\ e & \\
& \quad \mid \ e\ \texttt{match}\ \{\ [\texttt{case}\ x(x^*)\ \texttt{=>}\ e]^+\ \} \\
\text{Values} & v ::= \ldots \mid \langle x \rangle \mid \langle x \rangle (v^*) &
\end{array}
$$

$$\boxed{e \to e}$$

$$\frac{}{(\texttt{enum}\ t\ \{\ \overline{\texttt{case}\ x_i(\overline{\tau_{i,j}}^{\,j})}^{\,i}\ \};\ e) \to e[\overline{x_i \mapsto \langle x_i \rangle}^{\,i}]}$$

$$\frac{}{\langle x_k \rangle(\overline{v_{k,j}}^{\,j})\ \texttt{match}\ \{\ \overline{\texttt{case}\ x_i(\overline{x_{i,j}}^{\,j})\ \texttt{=>}\ e_i}^{\,i}\ \}) \to e_k[\overline{x_{k,j} \mapsto v_{k,j}}^{\,j}]}$$

$$E \quad ::= \quad \ldots \mid E\ \texttt{match}\ \{\ \overline{\texttt{case}\ x_i(\overline{x_{i,j}}^{\,j})\ \texttt{=>}\ e_i}^{\,i}\ \}$$

Algebraic Data Types

PLRG

Types	$\tau ::= \cdots \mid t$
Type Names	t

We need to check the **well-formedness** of types.

$$\boxed{\Gamma \vdash \tau}$$

$$\frac{}{\Gamma \vdash \texttt{num}} \qquad \frac{\Gamma \vdash \tau_1 \quad \ldots \quad \Gamma \vdash \tau_n \quad \Gamma \vdash \tau}{\Gamma \vdash (\tau_1, \ldots, \tau_n) \to \tau}$$

$$\frac{\Gamma(t) = x_i(\overline{\tau_{i,j}}^{\,j}) + \ldots + x_n(\overline{\tau_{n,j}}^{\,j})}{\Gamma \vdash t}$$

Algebraic Data Types

$$\boxed{\Gamma \vdash e : \tau}$$

$$\frac{t \notin \mathsf{dom}(\Gamma) \qquad \forall i, j.\ \Gamma' \vdash \tau_{i,j} \qquad \Gamma' = \Gamma[t = x_1(\overline{\tau_{1,j}}^{\,j}) + \ldots + x_n(\overline{\tau_{n,j}}^{\,j})] \qquad \Gamma'[\overline{x_i : (\overline{\tau_{i,j}}^{\,j}) \to t}^{\,i}] \vdash e : \tau \qquad \Gamma \vdash \tau}{\Gamma \vdash (\texttt{enum}\ t\ \{\ \overline{\texttt{case}\ x_i(\overline{\tau_{i,j}}^{\,j})}^{\,i}\ \};\ e) : \tau}$$

$$\frac{\Gamma \vdash e : t \qquad \Gamma(t) = x_1(\overline{\tau_{1,j}}^{\,j}) + \ldots + x_n(\overline{\tau_{n,j}}^{\,j}) \qquad \forall i.\ \Gamma[\overline{x_{i,j} : \tau_{i,j}}^{\,j}] \vdash e_i : \tau}{\Gamma \vdash (e\ \texttt{match}\ \{\ \overline{\texttt{case}\ x_i(\overline{x_{i,j}}^{\,j})\ \texttt{=>}\ e_i}^{\,i}\ \}) : \tau}$$

Contents

References

Let's extend our language with **references**.

$$
\begin{array}{lll}
\text{Expressions} & e & ::= \cdots \mid \texttt{ref}\ e \mid \texttt{!}e \mid e_1\ \texttt{:=}\ e_2 \mid a \\
\text{Values} & v & ::= \ldots \mid a \\
\text{Addresses} & a & \in \mathbb{A} \\
\text{Heaps} & h & \in \mathbb{A} \rightharpoonup \mathbb{V}
\end{array}
$$

$$\boxed{\langle h, e\rangle \to \langle h, e\rangle}$$

$$\frac{a \notin \mathsf{dom}(h)}{\langle h, \texttt{ref}\ v\rangle \to \langle h[a \mapsto v], a\rangle} \qquad \frac{h(a) = v}{\langle h, \texttt{!}a\rangle \to \langle h, v\rangle}$$

$$\frac{}{\langle h, a\ \texttt{:=}\ v\rangle \to \langle h[a \mapsto v], v\rangle}$$

$$E \quad ::= \quad \ldots \mid \texttt{ref}\ E \mid \texttt{!}E \mid E\ \texttt{:=}\ e \mid v\ \texttt{:=}\ E$$

References

Types $\tau ::= \cdots \mid \texttt{ref}\ \tau$

$$\boxed{\Gamma \vdash e : \tau}$$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \texttt{ref}\ e : \texttt{ref}\ \tau} \qquad \frac{\Gamma \vdash e : \texttt{ref}\ \tau}{\Gamma \vdash \texttt{!}e : \tau} \qquad \frac{\Gamma \vdash e_1 : \texttt{ref}\ \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ \texttt{:=}\ e_2 : \tau}$$

References

PLRG

$$\text{Types} \quad \tau ::= \cdots \mid \texttt{ref}\ \tau$$

$$\boxed{\Gamma \vdash e : \tau}$$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \texttt{ref}\ e : \texttt{ref}\ \tau} \qquad \frac{\Gamma \vdash e : \texttt{ref}\ \tau}{\Gamma \vdash \texttt{!}e : \tau} \qquad \frac{\Gamma \vdash e_1 : \texttt{ref}\ \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ \texttt{:=}\ e_2 : \tau}$$

However, it is **unsound** because there is no rule for typing **addresses**.

References

Let's fix the unsoundness by using **heap typing**.

$$\begin{array}{ll} \text{Types} & \tau ::= \cdots \mid \texttt{ref}\ \tau \\ \text{Store Typing} & \Sigma \in \mathbb{A} \rightharpoonup \mathbb{T} \end{array}$$

$$\boxed{\Gamma, \Sigma \vdash e : \tau}$$

$$\frac{\Gamma, \Sigma \vdash e : \tau}{\Gamma, \Sigma \vdash \texttt{ref}\ e : \texttt{ref}\ \tau} \qquad \frac{\Gamma, \Sigma \vdash e : \texttt{ref}\ \tau}{\Gamma, \Sigma \vdash \texttt{!}e : \tau}$$

$$\frac{\Gamma, \Sigma \vdash e_1 : \texttt{ref}\ \tau \qquad \Gamma, \Sigma \vdash e_2 : \tau}{\Gamma, \Sigma \vdash e_1\ \texttt{:=}\ e_2 : \tau} \qquad \frac{\Sigma(a) = \tau}{\Gamma, \Sigma \vdash a : \tau}$$

References

Definition (Well-Typed Heap)

A heap h is **well-typed** with respect to a store typing Σ, written $\Sigma \vdash h$, if for every $a \in \mathrm{dom}(h)$, we have $\Gamma, \Sigma \vdash h(a) : \Sigma(a)$.

References

Definition (Well-Typed Heap)

A heap h is **well-typed** with respect to a store typing Σ, written $\Sigma \vdash h$, if for every $a \in \mathrm{dom}(h)$, we have $\Gamma, \Sigma \vdash h(a) : \Sigma(a)$.

Lemma (Preservation)

If 1) $\Gamma, \Sigma \vdash e : \tau$*, 2)* $\Gamma, \Sigma \vdash h$*, and 3)* $\langle h, e \rangle \to \langle h', e' \rangle$*, then there exists* $\Sigma' \supseteq \Sigma$ *such that* $\Gamma, \Sigma' \vdash e' : \tau$ *and* $\Gamma, \Sigma' \vdash h'$*.*

References

PLRG

Definition (Well-Typed Heap)

A heap h is **well-typed** with respect to a store typing Σ, written $\Sigma \vdash h$, if for every $a \in \mathrm{dom}(h)$, we have $\Gamma, \Sigma \vdash h(a) : \Sigma(a)$.

Lemma (Preservation)

If 1) $\Gamma, \Sigma \vdash e : \tau$, *2)* $\Gamma, \Sigma \vdash h$, *and 3)* $\langle h, e\rangle \to \langle h', e'\rangle$, *then there exists* $\Sigma' \supseteq \Sigma$ *such that* $\Gamma, \Sigma' \vdash e' : \tau$ *and* $\Gamma, \Sigma' \vdash h'$.

Theorem (Soundness)

If 1) $\cdot, \Sigma \vdash e : \tau$, *2)* $\cdot, \Sigma \vdash h$, *3)* $\langle h, e\rangle \to^* \langle h', e'\rangle$, *and 4)* $\langle h', e'\rangle \not\to$, *then* e' *is a value and* $\cdot, \Sigma' \vdash e' : \tau$ *for some* $\Sigma' \supseteq \Sigma$.

Landin's Knot

Using mutable references, we can implement recursive functions without explicit recursion:

```
// A mutable box that can hold a function from Int to Int
case class Box(var content: Int => Int)

// Create a box that initially holds the identity function
val f = Box(x => x)

// Define the factorial function using the box
val factorial = (x: Int) => if (x == 0) 1 else x * f.content(x-1)

// Update the box to hold the factorial function
f.content = factorial

// Now we can call the factorial function
factorial(5) // == 5 * 4 * 3 * 2 * 1 == 120
```

Exceptions

Let's extend our language with **exceptions**.

$$\text{Expressions} \quad e ::= \cdots \mid \texttt{raise} \mid \texttt{try}\ e_1\ \texttt{catch}\ e_2$$

$$\boxed{e \to e}$$

$$\frac{}{E[\texttt{raise}] \to \texttt{raise}} \qquad \frac{e_1 \to e_1'}{\texttt{try}\ e_1\ \texttt{catch}\ e_2 \to \texttt{try}\ e_1'\ \texttt{catch}\ e_2}$$

$$\frac{}{\texttt{try raise catch}\ e_2 \to e_2} \qquad \frac{}{\texttt{try}\ v_1\ \texttt{catch}\ e_2 \to v_1}$$

Exceptions

There is no new types for exception.

$$\boxed{\Gamma \vdash e : \tau}$$

$$\frac{}{\Gamma \vdash \texttt{raise} : \tau} \qquad \frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \texttt{try}\ e_1\ \texttt{catch}\ e_2 : \tau}$$

Exceptions

Lemma (Progress)

If $\vdash e : \tau$, *then either*

- e *is a value,*
- e *is* `raise`, *or*
- *there exists* e' *such that* $e \to e'$.

Exceptions

PLRG

Lemma (Progress)

If $\vdash e : \tau$, *then either*

- e *is a value,*
- e *is* `raise`, *or*
- *there exists* e' *such that* $e \to e'$.

Theorem (Soundness)

If 1) $\vdash e : \tau$, *2)* $e \to^* e'$, *and 3)* $e' \not\to$, *then either*

- e' *is a value and* $\vdash e' : \tau$, *or*
- e' *is* `raise`.

Next Lecture

- Universal Types

Jihyeok Park
jihyeok_park@korea.ac.kr
https://plrg.korea.ac.kr