

Lecture 16 – Type Extensions

AAA551: Programming Language Theory

Jihyeok Park



2026 Spring

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

1. Algebraic Data Types

- Product Types

- Sum Types

- Algebraic Data Types

2. References and Exceptions

- References

- Exceptions

1. Algebraic Data Types

Product Types

Sum Types

Algebraic Data Types

2. References and Exceptions

References

Exceptions

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

Expressions $e ::= \dots \mid (e_1, e_2) \mid e.1 \mid e.2$

Values $v ::= \dots \mid (v_1, v_2)$

$$\boxed{e \rightarrow e}$$

$$\overline{(v_1, v_2).1 \rightarrow v_1} \quad \overline{(v_1, v_2).2 \rightarrow v_2}$$

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

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

$\Gamma \vdash e : \tau$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$
$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash e.1 : \tau_1} \quad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash e.2 : \tau_2}$$

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

Expressions $e ::= \dots \mid \text{inl } e \mid \text{inr } e$
 $\quad \quad \quad \mid \text{ case } e \text{ of inl } x \Rightarrow e_1 \mid \text{inr } y \Rightarrow e_2$
 Values $v ::= \dots \mid \text{inl } v \mid \text{inr } v$

$$\boxed{e \rightarrow e}$$

$$\frac{}{(\text{case } (\text{inl } v) \text{ of inl } x \Rightarrow e_1 \mid \text{inr } y \Rightarrow e_2) \rightarrow e_1[x \mapsto v]}$$

$$\frac{}{(\text{case } (\text{inr } v) \text{ of inl } x \Rightarrow e_1 \mid \text{inr } y \Rightarrow e_2) \rightarrow e_2[y \mapsto v]}$$

$E ::= \dots \mid \text{inl } E \mid \text{inr } E \mid \text{case } E \text{ of inl } x \Rightarrow e_1 \mid \text{inr } y \Rightarrow e_2$

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

$\Gamma \vdash e : \tau$

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

$$\frac{\Gamma \vdash e : \tau_1 + \tau_2 \quad \Gamma[x : \tau_1] \vdash e_1 : \tau \quad \Gamma[y : \tau_2] \vdash e_2 : \tau}{\Gamma \vdash (\mathbf{case} \ e \ \mathbf{of} \ \mathbf{inl} \ x \Rightarrow e_1 \mid \mathbf{inr} \ y \Rightarrow e_2) : \tau}$$

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

Expressions $e ::= \dots \mid \langle x \rangle$
 $\quad \quad \quad \mid \text{enum } t \{ [\text{case } x(\tau^*)]^+ \}; e$
 $\quad \quad \quad \mid e \text{ match } \{ [\text{case } x(x^*) \Rightarrow e]^+ \}$
 Values $v ::= \dots \mid \langle x \rangle \mid \langle x \rangle(v^*)$

$$\boxed{e \rightarrow e}$$

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

$$\frac{}{\langle x_k \rangle(\overline{v_{k,j}^j}) \text{ match } \{ \overline{\text{case } x_i(\overline{x_{i,j}^j}) \Rightarrow e_i^i} \} \rightarrow e_k[x_{k,j} \mapsto v_{k,j}^j]}$$

$$E ::= \dots \mid E \text{ match } \{ \overline{\text{case } x_i(\overline{x_{i,j}^j}) \Rightarrow e_i^i} \}$$

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

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

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

$$\frac{}{\Gamma \vdash \text{num}} \quad \frac{\Gamma \vdash \tau_1 \quad \dots \quad \Gamma \vdash \tau_n \quad \Gamma \vdash \tau}{\Gamma \vdash (\tau_1, \dots, \tau_n) \rightarrow \tau}$$

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

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

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

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

1. Algebraic Data Types

Product Types

Sum Types

Algebraic Data Types

2. References and Exceptions

References

Exceptions

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

Expressions $e ::= \dots \mid \mathbf{ref} \ e \mid !e \mid e_1 := e_2 \mid a$

Values $v ::= \dots \mid a$

Addresses $a \in \mathbb{A}$

Heaps $h \in \mathbb{A} \rightarrow \mathbb{V}$

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

$$\frac{a \notin \text{dom}(h)}{\langle h, \mathbf{ref} \ v \rangle \rightarrow \langle h[a \mapsto v], a \rangle} \qquad \frac{h(a) = v}{\langle h, !a \rangle \rightarrow \langle h, v \rangle}$$

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

$E ::= \dots \mid \mathbf{ref} \ E \mid !E \mid E := e \mid v := E$

Types $\tau ::= \dots \mid \text{ref } \tau$

$\Gamma \vdash e : \tau$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{ref } e : \text{ref } \tau} \quad \frac{\Gamma \vdash e : \text{ref } \tau}{\Gamma \vdash !e : \tau} \quad \frac{\Gamma \vdash e_1 : \text{ref } \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 := e_2 : \tau}$$

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

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

Types $\tau ::= \dots \mid \mathbf{ref} \tau$
 Store Typing $\Sigma \in \mathbb{A} \rightarrow \mathbb{T}$

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

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

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

Definition (Well-Typed Heap)

A heap h is **well-typed** with respect to a store typing Σ , written $\Sigma \vdash h$, if for every $a \in \text{dom}(h)$, 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) $\langle h, e \rangle \rightarrow \langle h', e' \rangle$, 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) $\langle h, e \rangle \rightarrow^ \langle h', e' \rangle$, and 4) $\langle h', e' \rangle \not\rightarrow$, then e' is a value and $\cdot, \Sigma' \vdash e' : \tau$ for some $\Sigma' \supseteq \Sigma$.*

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
```

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

Expressions $e ::= \dots \mid \text{raise} \mid \text{try } e_1 \text{ catch } e_2$

$$\boxed{e \rightarrow e}$$

$$\frac{}{E[\text{raise}] \rightarrow \text{raise}} \quad \frac{e_1 \rightarrow e'_1}{\text{try } e_1 \text{ catch } e_2 \rightarrow \text{try } e'_1 \text{ catch } e_2}$$

$$\frac{}{\text{try raise catch } e_2 \rightarrow e_2} \quad \frac{}{\text{try } v_1 \text{ catch } e_2 \rightarrow v_1}$$

There is no new types for exception.

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

$$\frac{}{\Gamma \vdash \text{raise} : \tau} \quad \frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{try } e_1 \text{ catch } e_2 : \tau}$$

Lemma (Progress)

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

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

Theorem (Soundness)

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

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

- Universal Types

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