Lecture 15 – Simply-Typed Lambda Calculus

AAA551: Programming Language Theory

Jihyeok Park

2026 Spring

Recall

- Motivation: Safe Language Systems
  - Detecting Run-Time Errors
  - Dynamic vs Static Analysis
  - Soundness vs Completeness
- Type System
  - Types
  - Type Errors
  - Type Checking
  - Type Soundness

Contents

Abstract Syntax

We can define the **abstract syntax** of **simply-typed lambda calculus** (STLC) as follows:

$$\begin{array}{ll} \text{Expressions} & e ::= n \mid x \mid e + e \mid \lambda x{:}\tau.\ e \mid e\ e \\ \text{Types} & \tau ::= \texttt{num} \mid \tau \to \tau \end{array}$$

We can define the abstract syntax of STLC in Scala as follows:

```
enum Expr:
  case Num(num: Int)
  case Var(name: String)
  case Add(left: Expr, right: Expr)
  case Abs(param: String, ty: Type, body: Expr)
  case App(fun: Expr, arg: Expr)

enum Type:
  case NumT
  case ArrowT(paramTy: Type, retTy: Type)
```

Dynamic Semantics

PLRG

Expressions	$e ::= n \mid x \mid e + e \mid \lambda x : \tau.\ e \mid e\ e$
Types	$\tau ::= \texttt{num} \mid \tau \to \tau$
Values	$v ::= \lambda x : \tau.\ e \mid n$

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

$$\frac{n = n_1 + n_2}{n_1 + n_2 \to n}\ \textsc{Add} \qquad \frac{}{(\lambda x : \tau.\ e)(v) \to e[x \mapsto v]}\ \beta\textsc{-Reduce}$$

$$\frac{e_1 \to e_2}{E[e_1] \to E[e_2]}\ \textsc{Context}$$

$$E ::= [\cdot] \mid E\ e \mid v\ E \mid E + e \mid v + E$$

Static Semantics

Expressions	$e ::= n \mid x \mid e + e \mid \lambda x{:}\tau.\ e \mid e\ e$
Types	$\tau ::= \mathtt{num} \mid \tau \to \tau$
Values	$v ::= \lambda x{:}\tau.\ e \mid n$
Typing Environments	$\Gamma \in \mathbb{X} \rightharpoonup \tau$

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

$$\frac{}{\Gamma \vdash n : \mathtt{num}}\ \textsc{T-Num} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \textsc{T-Var}$$

$$\frac{\Gamma \vdash e_1 : \mathtt{num} \qquad \Gamma \vdash e_2 : \mathtt{num}}{\Gamma \vdash e_1 + e_2 : \mathtt{num}}\ \textsc{T-Add}$$

$$\frac{\Gamma[x : \tau] \vdash e : \tau'}{\Gamma \vdash \lambda x{:}\tau.\ e : \tau \to \tau'}\ \textsc{T-Abs} \qquad \frac{\Gamma \vdash e_1 : \tau \to \tau' \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ e_2 : \tau'}\ \textsc{T-App}$$

Static Semantics

```
def tycheck(expr: Expr, tenv: Map[String, Type]): Type = expr match
  case Num(_) => NumT
  case Var(name) => tenv.getOrElse(name, error(s"unbound id: \$name"))
  case Add(left, right) =>
    mustEqual(tycheck(left, tenv), NumT)
    mustEqual(tycheck(right, tenv), NumT)
    NumT
  case Abs(param, ty, body) =>
    val bodyTy = tycheck(body, tenv + (param -> ty))
    ArrowT(ty, bodyTy)
  case App(fun, arg) =>
    tycheck(fun, tenv) match
      case ArrowT(paramTy, retTy) =>
        mustEqual(tycheck(arg, tenv), paramTy)
        retTy
      case actual => error(s"arrow type expected, but got \$actual")

def mustEqual(actual: Type, expected: Type): Unit =
  if (expected != actual) error(s"$expected expected, but got $actual")
```

Static Semantics – Examples

$$\dfrac{\dfrac{\dfrac{}{\Gamma_0 \vdash \mathtt{x} : \mathtt{num}} \quad \dfrac{}{\Gamma_0 \vdash 40 : \mathtt{num}}}{\Gamma_0 \vdash \mathtt{x} + 40 : \mathtt{num}}}{\dfrac{\vdash \lambda \mathtt{x} : \mathtt{num}.\ \mathtt{x} + 40 : \mathtt{num} \to \mathtt{num}}{\vdash (\lambda \mathtt{x} : \mathtt{num}.\ \mathtt{x} + 40)(2)} \quad \dfrac{}{\vdash 2 : \mathtt{num}}}$$

where $\Gamma_0 = [\mathtt{x} : \mathtt{num}]$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_0 \vdash \mathtt{f} : \mathtt{num} \to \mathtt{num}} \quad \dfrac{}{\Gamma_0 \vdash 2 : \mathtt{num}}}{\Gamma_0 \vdash \mathtt{f}(2) : \mathtt{num}}}{\vdash \lambda \mathtt{f} : \mathtt{num} \to \mathtt{num}.\ \mathtt{f}(2) : (\mathtt{num} \to \mathtt{num}) \to \mathtt{num}} \quad \dfrac{\dfrac{}{\Gamma_1 \vdash \mathtt{x} : \mathtt{num}}}{\vdash \lambda \mathtt{x} : \mathtt{num}.\ \mathtt{x} : \mathtt{num} \to \mathtt{num}}}{\vdash (\lambda \mathtt{f} : \mathtt{num} \to \mathtt{num}.\ \mathtt{f}(2))(\lambda \mathtt{x} : \mathtt{num}.\ \mathtt{x}) : \mathtt{num}}$$

where $\Gamma_0 = [\mathtt{f} : \mathtt{num} \to \mathtt{num}]$ and $\Gamma_1 = [\mathtt{x} : \mathtt{num}]$.

Contents

Type Soundness

PLRG

Definition (Type Soundness)

A **type system** is **sound** if it guarantees that a **well-typed** program will **never** cause a **type error** at run-time.

Type Soundness

PLRG

Theorem (Type Soundness)

The STLC is type sound:

$$(\vdash e : \tau \land e \to^* e' \land e' \not\to) \implies (e' \text{ is a value and } \vdash e' : \tau)$$

Lemma (Progress)

If an expression is well-typed, then it is either a value or it can take a small-step evaluation step:

$$(\vdash e : \tau) \implies (e \text{ is a value or } \exists e'.\ e \to e')$$

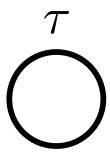
Lemma (Preservation)

If a well-typed expression has a small-step evaluation step, then the resulting expression is also well-typed with the same type:

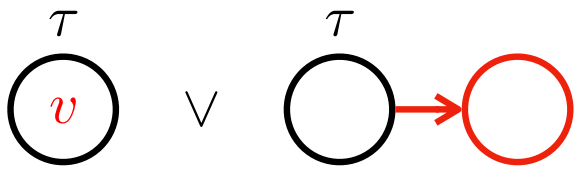
$$(\vdash e : \tau \land e \to e') \implies (\vdash e' : \tau)$$

Soundness

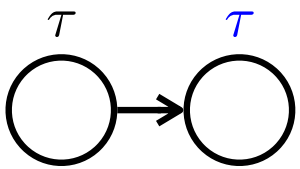
$$(\vdash e : \tau \land e \to^* e' \land e' \not\to) \implies (e' \text{ is a value and } \vdash e' : \tau)$$



Progress

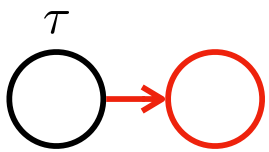


Preservation

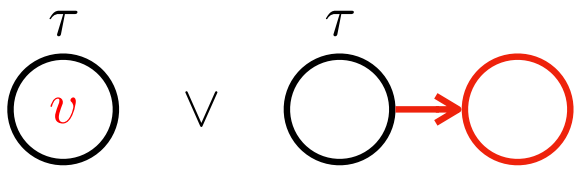


Soundness

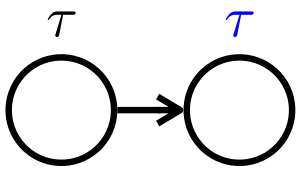
$$(\vdash e : \tau \land e \to^* e' \land e' \not\to) \implies (e' \text{ is a value and } \vdash e' : \tau)$$



Progress

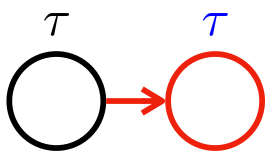


Preservation

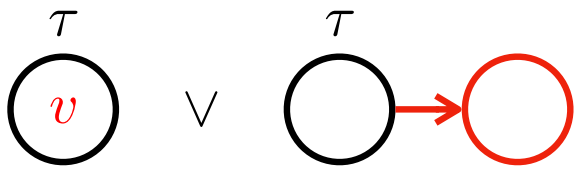


Soundness

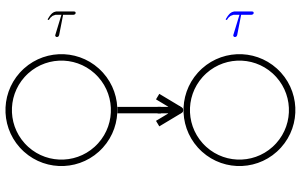
$$(\vdash e : \tau \land e \to^* e' \land e' \not\to) \implies (e' \text{ is a value and } \vdash e' : \tau)$$



Progress

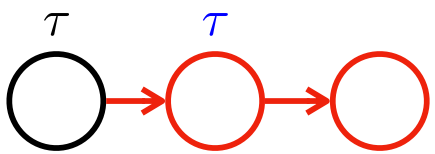


Preservation

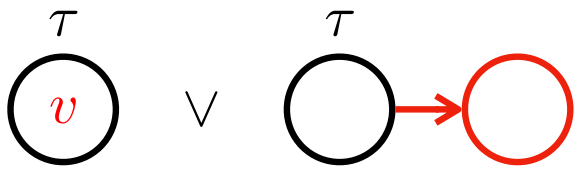


Soundness

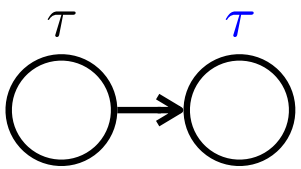
$$(\vdash e : \tau \land e \to^* e' \land e' \not\to) \implies (e' \text{ is a value and } \vdash e' : \tau)$$



Progress

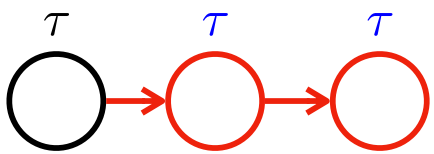


Preservation

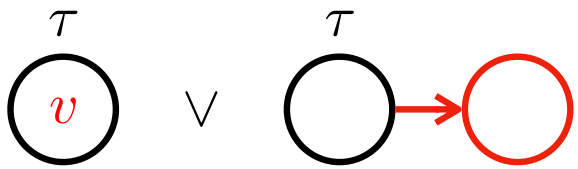


Soundness

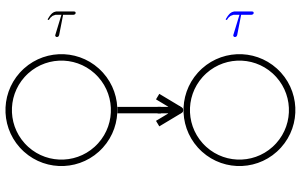
$$(\vdash e : \tau \land e \to^* e' \land e' \not\to) \implies (e' \text{ is a value and } \vdash e' : \tau)$$



Progress

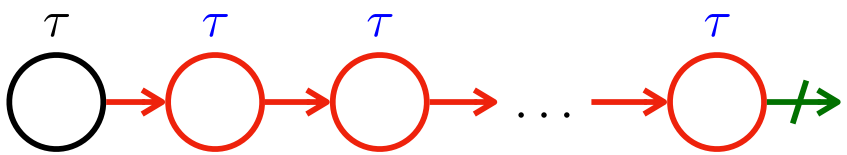


Preservation

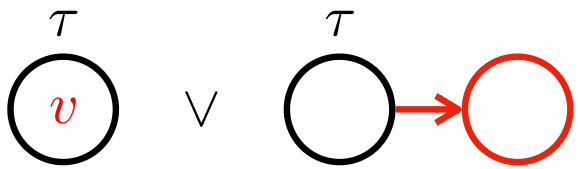


Soundness

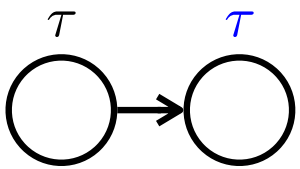
$$(\vdash e : \tau \land e \to^* e' \land e' \not\to) \implies (e' \text{ is a value and } \vdash e' : \tau)$$



Progress

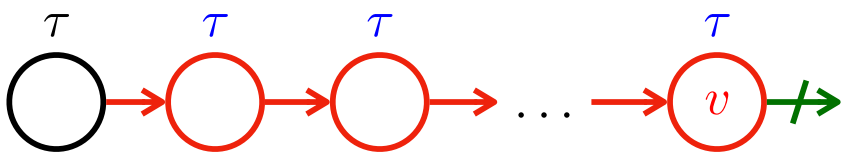


Preservation

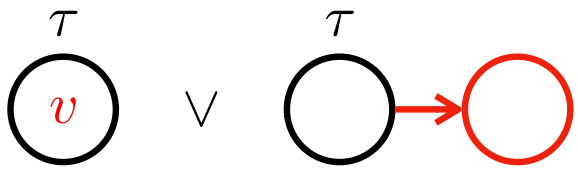


Soundness

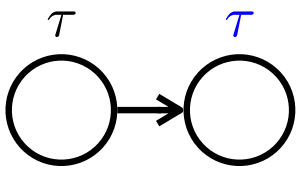
$$(\vdash e : \tau \land e \to^* e' \land e' \not\to) \implies (e' \text{ is a value and } \vdash e' : \tau)$$



Progress



Preservation



Progress

Lemma (Canonical Forms)

If a value v has a type τ (i.e., $\vdash v : \tau$), then:

1. *If $\tau = \mathtt{num}$, then $v = n$ for some n.*
2. *If $\tau = \tau_1 \to \tau_2$, then $v = \lambda x{:}\tau_1.e$ for some x and e.*

Using the canonical forms lemma, we can prove the progress lemma:

Lemma (Progress)

If an expression is well-typed, then it is either a value or it can take a small-step evaluation step:

$$(\vdash e : \tau) \implies (e \text{ is a value or } \exists e'.\ e \to e')$$

Let's prove the progress lemma by induction on the structure of typing rules.

Progress

- T-Num and T-Abs cases are trivial since they are values.

$$\frac{}{\Gamma \vdash n : \texttt{num}}\ \text{T-Num} \qquad \frac{\Gamma[x:\tau] \vdash e : \tau'}{\Gamma \vdash \lambda x{:}\tau.\ e : \tau \to \tau'}\ \text{T-Abs}$$

- T-Var case is impossible since variables with an empty typing environment cannot be well-typed.

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \text{T-Var}$$

Progress

- T-ADD case. $\vdash e_1 \texttt{ + } e_2 : \texttt{num}$ implies $\vdash e_1 : \texttt{num}$ and $\vdash e_2 : \texttt{num}$.

$$\frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash e_1 \texttt{ + } e_2 : \texttt{num}} \text{ T-ADD}$$

By induction hypothesis, each e_i is either 1) a value (i.e., $e_i = v_i$) or 2) can take a step (i.e., $e_i \to e_i'$).

- If e_1 is not a value, by the CONTEXT rule,

$$e_1 \texttt{ + } e_2 \to e_1' \texttt{ + } e_2 \qquad (E = [\cdot] \texttt{ + } e_2)$$

- If e_1 is a value but e_2 is not a value, by the CONTEXT rule,

$$v_1 \texttt{ + } e_2 \to v_1 \texttt{ + } e_2' \qquad (E = v_1 \texttt{ + } [\cdot])$$

- If e_1 and e_2 are both values, then by the canonical forms lemma and $\vdash e_i : \texttt{num}$, they must be numbers ($e_1 = n_1$ and $e_2 = n_2$). So, we can take a step using the ADD rule:

$$n_1 \texttt{ + } n_2 \to n \qquad (n = n_1 + n_2)$$

Progress

- T-App case. $\vdash e_1\ e_2 : \tau$ implies $\vdash e_1 : \tau' \to \tau$ and $\vdash e_2 : \tau'$.

$$\frac{\Gamma \vdash e_1 : \tau' \to \tau \qquad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash e_1\ e_2 : \tau}\ \text{T-App}$$

By induction hypothesis, each e_i is either 1) a value (i.e., $e_i = v_i$) or 2) can take a step (i.e., $e_i \to e_i'$).

- If e_1 is not a value, by the Context rule,

$$e_1\ e_2 \to e_1'(e_2) \qquad (E = [\cdot]\ e_2)$$

- If e_1 is a value but e_2 is not a value, by the Context rule,

$$v_1\ e_2 \to v\ e_2' \qquad (E = v\ [\cdot])$$

- If e_1 and e_2 are both values, then by the canonical forms lemma and $\vdash e_1 : \tau' \to \tau$, e_1 must be a lambda abstraction (i.e., $e_1 = \lambda x{:}\tau'.e$). So, we can take a step using the β-Reduce rule:

$$(\lambda x{:}\tau'.e)(v) \to e[x \mapsto v]$$

Preservation

PLRG

Lemma (Substitution)

If $[x : \tau] \vdash e : \tau'$ *and* $\vdash v : \tau$, *then* $\vdash e[x \mapsto v] : \tau'$.

Lemma (Context)

If $\vdash E[e] : \tau$ *and* $\vdash e : \tau'$, *then* $\vdash E[e'] : \tau$ *for any* e' *such that* $\vdash e' : \tau'$.

We will skip the detailed proof of above lemmas, but they can be proved by induction on the structure of typing rules.

Lemma (Preservation)

If a well-typed expression has a small-step evaluation step, then the resulting expression is also well-typed with the same type:

$$(\vdash e : \tau \land e \to e') \implies (\vdash e' : \tau)$$

Preservation

- ADD case.

$$\frac{n = n_1 + n_2}{n_1 \texttt{ + } n_2 \to n}\ \textsc{Add}$$

n_1 + n_2 has type `num` by the T-ADD and T-NUM rules:

$$\frac{\dfrac{}{\vdash n_1 : \texttt{num}}\ \textsc{T-Num} \qquad \dfrac{}{\vdash n_2 : \texttt{num}}\ \textsc{T-Num}}{\vdash n_1 \texttt{ + } n_2 : \texttt{num}}\ \textsc{T-Add}$$

Similarly, n has type `num` by the T-NUM rule:

$$\frac{}{\vdash n : \texttt{num}}\ \textsc{T-Num}$$

So, the preservation property holds by preserving the type `num`.

Preservation

- β-Reduce case.

$$\frac{}{(\lambda x : \tau.\ e)(v) \to e[x \mapsto v]}\ \beta\text{-Reduce}$$

$(\lambda x : \tau.\ e)(v)$ is well-typed. Let's assume that it has type τ'.

Then, we have the following typing derivation:

$$\frac{\dfrac{[x : \tau] \vdash e : \tau'}{\vdash \lambda x : \tau.\ e : \tau \to \tau'}\ \text{T-Abs} \qquad \dfrac{\ldots}{\vdash v : \tau}}{\vdash (\lambda x : \tau.\ e)(v) : \tau'}\ \text{T-App}$$

By the substitution lemma, $[x : \tau] \vdash e : \tau'$ and $\vdash v : \tau$ imply:

$$\vdash e[x \mapsto v] : \tau'$$

So, the preservation property holds by preserving the type τ'.

Preservation

- Context case.

$$\frac{e_1 \to e_2}{E[e_1] \to E[e_2]}\ \textsc{Context}$$

$$E \quad ::= \quad [\cdot] \mid E\ e \mid v\ E \mid E + e \mid v + E$$

$E[e_1]$ is well-typed. Let's assume that it has type τ.

By the induction on the structure of evaluation contexts, we can show that e_1 is well-typed with some type τ_1 (i.e., $\vdash e_1 : \tau_1$).

By the induction hypothesis and $e_1 \to e_2$, we have $\vdash e_2 : \tau_1$.

By the context lemma, $\vdash E[e_1] : \tau$ and $\vdash e_1 : \tau_1$ imply:

$$\vdash E[e_2] : \tau$$

So, the preservation property holds by preserving the type τ.

Next Lecture

- Type Extensions

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