Lecture 18 – Subtyping

AAA551: Programming Language Theory

Jihyeok Park

2026 Spring

Recall

PLRG

- Polymorphism
- Universal Types
  - Syntax and Dynamic Semantics
  - Well-Formedness
  - Type Equivalence
  - Typing Rules
  - Type Erasure
  - Examples
  - Normalization

Contents

PLRG

1. Subtyping

2. STLC with Subtyping
   - Typing Rules
   - Subtype Relation
   - Variance
   - Algorithmic Typing Rules

3. Other Extensions
   - Exceptions
   - Records
   - Conditional Expressions

4. Bounded Quantification

Contents

1. Subtyping

2. STLC with Subtyping
   - Typing Rules
   - Subtype Relation
   - Variance
   - Algorithmic Typing Rules

3. Other Extensions
   - Exceptions
   - Records
   - Conditional Expressions

4. Bounded Quantification

Recall: Polymorphism

PLRG

A **polymorphism** is to allow **multiple types** for a single term.

There are three representative polymorphisms:

- **Subtype polymorphism** defines a subtype relation between types, and allows a term of a type to be used as a term of its supertypes.
- **Ad-hoc polymorphism** allows different implementations for the same function name with different types (e.g., overloading, type classes).
- **Parametric polymorphism** defines a term with type variables, and allows the term to be instantiated with different types.

Recall: Polymorphism

PLRG

A **polymorphism** is to allow **multiple types** for a single term.

There are three representative polymorphisms:

- **Subtype polymorphism** defines a subtype relation between types, and allows a term of a type to be used as a term of its supertypes.
- **Ad-hoc polymorphism** allows different implementations for the same function name with different types (e.g., overloading, type classes).
- **Parametric polymorphism** defines a term with type variables, and allows the term to be instantiated with different types.

In this lecture, we will focus on **subtype polymorphism**.

Subtyping

PLRG

Subtyping is a relation between types, denoted as $S <: T$, which means that type S is a **subtype** of type T.

Subtyping

PLRG

Subtyping is a relation between types, denoted as $S <: T$, which means that type S is a **subtype** of type T.

If $S <: T$, then a term of type S can be used in any context where a term of type T is expected.

Subtyping

PLRG

Subtyping is a relation between types, denoted as $S <: T$, which means that type S is a **subtype** of type T.

If $S <: T$, then a term of type S can be used in any context where a term of type T is expected.

For example, if we have a type hierarchy where `Dog` $<:$ `Animal`, then we can use a term of type `Dog` wherever a term of type `Animal` is expected.

```
trait Animal:
  def speak: String

case class Dog(name: String) extends Animal:
  def speak: String = name + " says woof."

val dog: Dog = Dog("Buddy")
val animal: Animal = dog // Upcasting from Dog to Animal
animal.speak             // Output: "Buddy says woof."
```

Contents

1. Subtyping

2. STLC with Subtyping
   - Typing Rules
   - Subtype Relation
   - Variance
   - Algorithmic Typing Rules

3. Other Extensions
   - Exceptions
   - Records
   - Conditional Expressions

4. Bounded Quantification

Typing Rules

PLRG

Expressions $e ::= n \mid x \mid e \texttt{ + } e \mid \lambda x{:}\tau.\ e \mid e\ e$

Types $\tau ::= \texttt{num} \mid \tau \to \tau \mid \bot \mid \top$

Typing Rules

PLRG

Expressions $e ::= n \mid x \mid e + e \mid \lambda x{:}\tau.\ e \mid e\ e$

Types $\tau ::= \texttt{num} \mid \tau \to \tau \mid \bot \mid \top$

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

$$\frac{}{\Gamma \vdash n : \texttt{num}} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash e_1 + e_2 : \texttt{num}}$$

$$\frac{\Gamma[x : \tau] \vdash e : \tau'}{\Gamma \vdash \lambda x{:}\tau.\ e : \tau \to \tau'} \qquad \frac{\Gamma \vdash e_1 : \tau \to \tau' \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ e_2 : \tau'}$$

We have one more rule called the **subsumption** rule for subtyping:

$$\frac{\Gamma \vdash e : \tau \qquad \tau <: \tau'}{\Gamma \vdash e : \tau'}$$

Subtype Relation

PLRG

Expressions	$e ::= n \mid x \mid e + e \mid \lambda x{:}\tau.\ e \mid e\ e$
Types	$\tau ::= \mathtt{num} \mid \tau \to \tau \mid \bot \mid \top$

$$\boxed{\tau <: \tau}$$

$$\frac{}{\tau <: \tau} \qquad \frac{\tau_1 <: \tau_2 \qquad \tau_2 <: \tau_3}{\tau_1 <: \tau_3} \qquad \frac{}{\bot <: \tau} \qquad \frac{}{\tau <: \top}$$

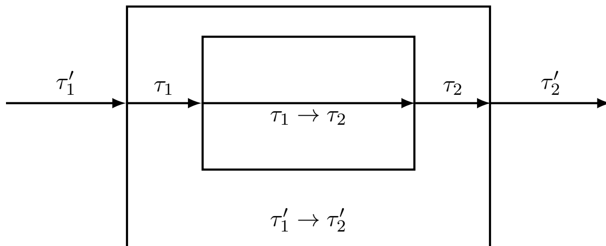
$$\frac{\tau_1' <: \tau_1 \qquad \tau_2 <: \tau_2'}{\tau_1 \to \tau_2 <: \tau_1' \to \tau_2'}$$

```
val x: Any = 42                    // Int              <: Any (Top)
def f(x: Nothing): String = x // Nothing (Bottom) <: String
```

Subtype Relation – Function Types

PLRG

$$\frac{\tau_1' <: \tau_1 \qquad \tau_2 <: \tau_2'}{\tau_1 \to \tau_2 <: \tau_1' \to \tau_2'}$$




```
val f: Any => String = (x: Any) => x.toString
val g: Int => Any = f // (Any => String) <: (Int => Any)
```

Variance

PLRG

In general, a type constructor can be:

- **Covariant** if it preserves the subtype relation.
- **Contravariant** if it reverses the subtype relation.
- **Invariant** if it does not preserve or reverse the subtype relation.

Variance

PLRG

In general, a type constructor can be:

- **Covariant** if it preserves the subtype relation.
- **Contravariant** if it reverses the subtype relation.
- **Invariant** if it does not preserve or reverse the subtype relation.

$$\frac{\tau_1' <: \tau_1 \qquad \tau_2 <: \tau_2'}{\tau_1 \to \tau_2 <: \tau_1' \to \tau_2'}$$

The subtype relation for function types is called:

- **contravariant** in the parameter type and
- **covariant** in the return type.

Variance

In Scala, we can define type parameters with variance annotations:

```scala
class X[+A] // Covariant type parameter
class Y[-A] // Contravariant type parameter
class Z[A]  // Invariant type parameter

val intX: X[Int] = new X[Int]
val botX: X[Nothing] = intX       // Error: X[Int] !<: X[Nothing]
val anyX: X[Any] = intX           // OK    : X[Int] <: X[Any]

val intY: Y[Int] = new Y[Int]
val botY: Y[Nothing] = intY       // OK    : Y[Int] <: Y[Nothing]
val anyY: Y[Any] = intY           // Error: Y[Int] !<: Y[Any]

val intZ: Z[Int] = new Z[Int]
val botZ: Z[Nothing] = intZ       // Error: Z[Int] !<: Z[Nothing]
val anyZ: Z[Any] = intZ           // Error: Z[Int] !<: Z[Any]
```

Variance – Lists vs. Arrays

PLRG

In Scala language, **immutable** `List` is a **covariant** type constructor, while **mutable** `Array` is an **invariant** type constructor. Why?

$$\frac{\tau_1 <: \tau_2}{\texttt{List}[\tau_1] <: \texttt{List}[\tau_2]} \qquad \frac{}{\texttt{Array}[\tau] <: \texttt{Array}[\tau]}$$

Variance – Lists vs. Arrays

PLRG

In Scala language, **immutable** `List` is a **covariant** type constructor, while **mutable** `Array` is an **invariant** type constructor. Why?

$$\frac{\tau_1 <: \tau_2}{\texttt{List}[\tau_1] <: \texttt{List}[\tau_2]} \qquad \frac{}{\texttt{Array}[\tau] <: \texttt{Array}[\tau]}$$

The covariance of `List` is safe:

```
val intList: List[Int] = List(1, 2, 3)
val anyList: List[Any] = intList          // List[Int] <: List[Any]
val x: Int = intList.head                 // 1
x + 1                                     // OK
```

Variance – Lists vs. Arrays

PLRG

In Scala language, **immutable** `List` is a **covariant** type constructor, while **mutable** `Array` is an **invariant** type constructor. Why?

$$\frac{\tau_1 <: \tau_2}{\texttt{List}[\tau_1] <: \texttt{List}[\tau_2]} \qquad \frac{}{\texttt{Array}[\tau] <: \texttt{Array}[\tau]}$$

The covariance of `List` is safe:

```
val intList: List[Int] = List(1, 2, 3)
val anyList: List[Any] = intList          // List[Int] <: List[Any]
val x: Int = intList.head                 // 1
x + 1                                     // OK
```

If we assume that `Array` is covariant, it would be unsafe:

```
val intArray: Array[Int] = Array(1, 2, 3)
val anyArray: Array[Any] = intArray       // Array[Int] <: Array[Any]
anyArray(0) = "Hello"                     // String <: Any
val y: Int = intArray(0)                  // "Hello" is not an Int!
y + 1                                     // Runtime error: "Hello" + 1
```

Algorithmic Typing Rules

PLRG

$$\frac{\Gamma \vdash e : \tau \qquad \tau <: \tau'}{\Gamma \vdash e : \tau'}$$

The subsumption rule is **non-syntax-directed** because τ' is not determined by the syntax of e.

Algorithmic Typing Rules

PLRG

$$\frac{\Gamma \vdash e : \tau \qquad \tau <: \tau'}{\Gamma \vdash e : \tau'}$$

The subsumption rule is **non-syntax-directed** because τ' is not determined by the syntax of e.

Let's define an **algorithmic** typing rule without the subsumption rule:

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

$$\frac{}{\Gamma \vdash n : \texttt{num}} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \tau_1 <: \texttt{num} \qquad \Gamma \vdash e_2 : \tau_2 \qquad \tau_2 <: \texttt{num}}{\Gamma \vdash e_1 \texttt{ + } e_2 : \texttt{num}}$$

$$\frac{\Gamma[x : \tau] \vdash e : \tau'}{\Gamma \vdash \lambda x{:}\tau.\ e : \tau \to \tau'} \qquad \frac{\Gamma \vdash e_1 : \tau \to \tau' \qquad \Gamma \vdash e_2 : \tau'' \qquad \tau'' <: \tau}{\Gamma \vdash e_1\ e_2 : \tau'}$$

Contents

PLRG

1. Subtyping

2. STLC with Subtyping
    Typing Rules
    Subtype Relation
    Variance
    Algorithmic Typing Rules

3. Other Extensions
    Exceptions
    Records
    Conditional Expressions

4. Bounded Quantification

Exceptions

PLRG

$$\text{Expressions} \quad e ::= \ldots \mid \texttt{raise}$$

$$\frac{}{E[\texttt{raise}] \to \texttt{raise}}$$

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

$$\frac{}{\Gamma \vdash \texttt{raise} : \bot}$$

```
def f: Int = throw new Error       // Nothing <: Int
def g: Boolean = throw new Error  // Nothing <: Boolean
def h: String = throw new Error   // Nothing <: String
```

Records

Expressions	$e ::= \ldots \mid \{x_1\texttt{=}e_1,\ \ldots,\ x_n\texttt{=}e_n\} \mid e.x$
Values	$v ::= \ldots \mid \{x_1\texttt{=}v_1,\ \ldots,\ x_n\texttt{=}v_n\}$
Types	$\tau ::= \ldots \mid \{x_1 : \tau_1,\ \ldots,\ x_n : \tau_n\}$

Records

PLRG

$$
\begin{array}{lll}
\text{Expressions} & e ::= \ldots \mid \{x_1\texttt{=}e_1,\ \ldots,\ x_n\texttt{=}e_n\} \mid e.x \\
\text{Values} & v ::= \ldots \mid \{x_1\texttt{=}v_1,\ \ldots,\ x_n\texttt{=}v_n\} \\
\text{Types} & \tau ::= \ldots \mid \{x_1 : \tau_1,\ \ldots,\ x_n : \tau_n\}
\end{array}
$$

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

$$\frac{}{\{x_1\texttt{=}v_1,\ \ldots,\ x_n\texttt{=}v_n\}.x_i \to v_i}$$

$$E \quad ::= \quad \ldots \mid \{x_1\texttt{=}E,\ \ldots,\ x_n\texttt{=}e_n\} \mid \ldots \mid \{x_1\texttt{=}v_1,\ \ldots,\ x_n\texttt{=}E\} \mid E.x$$

Records

Expressions	$e ::= \ldots \mid \{x_1\texttt{=}e_1,\ \ldots,\ x_n\texttt{=}e_n\} \mid e.x$
Values	$v ::= \ldots \mid \{x_1\texttt{=}v_1,\ \ldots,\ x_n\texttt{=}v_n\}$
Types	$\tau ::= \ldots \mid \{x_1 : \tau_1,\ \ldots,\ x_n : \tau_n\}$

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

$$\frac{}{\{x_1\texttt{=}v_1,\ \ldots,\ x_n\texttt{=}v_n\}.x_i \rightarrow v_i}$$

$$E \quad ::= \quad \ldots \mid \{x_1\texttt{=}E,\ \ldots,\ x_n\texttt{=}e_n\} \mid \ldots \mid \{x_1\texttt{=}v_1,\ \ldots,\ x_n\texttt{=}E\} \mid E.x$$

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

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \ldots \qquad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash \{x_1\texttt{=}e_1,\ \ldots,\ x_n\texttt{=}e_n\} : \{x_1 : \tau_1,\ \ldots,\ x_n : \tau_n\}} \qquad \frac{\Gamma \vdash e : \{\ldots,\ x_i : \tau_i,\ \ldots\}}{\Gamma \vdash e.x_i : \tau_i}$$

Subtyping for Records

PLRG

$$\text{Types} \quad \tau ::= \ldots \mid \{x_1 : \tau_1,\ \ldots,\ x_n : \tau_n\}$$

$$\boxed{\tau <: \tau}$$

$$\frac{\tau_1 <: \tau_1' \qquad \ldots \qquad \tau_n <: \tau_n'}{\{x_1 : \tau_1,\ \ldots,\ x_n : \tau_n\} <: \{x_1 : \tau_1',\ \ldots,\ x_n : \tau_n'\}}$$

$$\frac{}{\{x_1 : \tau_1,\ \ldots,\ x_n : \tau_n, x_{n+1} : \tau_{n+1}\} <: \{x_1 : \tau_1,\ \ldots,\ x_n : \tau_n\}}$$

$$\frac{\{k_1, \ldots, k_n\} \text{ is a permutation of } \{1, \ldots, n\}}{\{x_1 : \tau_1,\ \ldots,\ x_n : \tau_n\} <: \{x_{k_1} : \tau_{k_1},\ \ldots,\ x_{k_n} : \tau_{k_n}\}}$$

Conditional Expressions

PLRG

Expressions	$e ::= \cdots \mid \texttt{true} \mid \texttt{false} \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e$
Values	$v ::= \ldots \mid \texttt{true} \mid \texttt{false}$
Types	$\tau ::= \ldots \mid \texttt{bool}$

Conditional Expressions

PLRG

$$
\begin{array}{lrcl}
\text{Expressions} & e & ::= & \cdots \mid \texttt{true} \mid \texttt{false} \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \\
\text{Values} & v & ::= & \ldots \mid \texttt{true} \mid \texttt{false} \\
\text{Types} & \tau & ::= & \ldots \mid \texttt{bool}
\end{array}
$$

$$\boxed{\tau <: \tau}$$

We can use the following rule with the subsumption rule:

$$\frac{}{\Gamma \vdash \texttt{true} : \texttt{bool}} \qquad \frac{}{\Gamma \vdash \texttt{false} : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \tau}$$

Conditional Expressions

PLRG

$$
\begin{array}{lrcl}
\text{Expressions} & e & ::= & \cdots \mid \texttt{true} \mid \texttt{false} \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \\
\text{Values} & v & ::= & \ldots \mid \texttt{true} \mid \texttt{false} \\
\text{Types} & \tau & ::= & \ldots \mid \texttt{bool}
\end{array}
$$

$$\boxed{\tau <: \tau}$$

We can use the following rule with the subsumption rule:

$$\frac{}{\Gamma \vdash \texttt{true} : \texttt{bool}} \qquad \frac{}{\Gamma \vdash \texttt{false} : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \tau}$$

Then, how to define algorithmic typing rules without the subsumption rule?

Conditional Expressions – Join and Meet

PLRG

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \qquad \Gamma \vdash e_2 : \tau_2 \qquad \Gamma \vdash e_3 : \tau_3}{\Gamma \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \tau_2 \sqcup \tau_3}$$

where $\tau \sqcup \tau'$ (or $\tau \sqcap \tau'$) is the least upper bound (or greatest lower bound) of τ and τ' with respect to the subtype relation:

$$\tau \sqcup \tau' = \begin{cases} \tau' & \text{if } \tau <: \tau' \\ \tau & \text{if } \tau :> \tau' \\ (\tau_1 \sqcap \tau_1') \to (\tau_2 \sqcup \tau_2') & \text{if } \tau = \tau_1 \to \tau_2 \text{ and } \tau' = \tau_1' \to \tau_2' \\ \top & \text{otherwise} \end{cases}$$

$$\tau \sqcap \tau' = \begin{cases} \tau & \text{if } \tau <: \tau' \\ \tau' & \text{if } \tau :> \tau' \\ (\tau_1 \sqcup \tau_1') \to (\tau_2 \sqcap \tau_2') & \text{if } \tau = \tau_1 \to \tau_2 \text{ and } \tau' = \tau_1' \to \tau_2' \\ \bot & \text{otherwise} \end{cases}$$

Conditional Expressions – Union and Intersection

PLRG

Scala supports union types (`A | B`) and intersection types (`A & B`) between types `A` and `B`.

Using union and intersection types, Scala supports a more precise type for the conditional expression:

```
if (true) (x: Int) => x else (y: String) => y
// (Int & String) => (Int | String)
```

Without union and intersection types, the type of the above expression would be `Nothing => Any`, which is less precise than `(Int & String) => (Int | String)`.

Contents

PLRG

1. Subtyping

2. STLC with Subtyping
   - Typing Rules
   - Subtype Relation
   - Variance
   - Algorithmic Typing Rules

3. Other Extensions
   - Exceptions
   - Records
   - Conditional Expressions

4. Bounded Quantification

Kernel $F_{<:}$

PLRG

We can extend the **system** F with subtyping, called **kernel** $F_{<:}$, by adding **bounded quantification**:

$$\begin{array}{llcl} \text{Expressions} & e & ::= & n \mid x \mid e + e \mid \lambda x : \tau .\ e \mid e\ e \mid \Lambda\alpha <: \tau .\ e \mid e[\tau] \\ \text{Types} & \tau & ::= & \texttt{num} \mid \tau \to \tau \mid \alpha \mid \forall\alpha <: \tau .\tau \end{array}$$

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

$$\frac{\alpha \notin \mathsf{dom}(\Gamma) \qquad \Gamma \vdash \tau \qquad \Gamma[\alpha <: \tau] \vdash e : \tau'}{\Gamma \vdash \Lambda\alpha <: \tau .\ e : \forall\alpha <: \tau .\tau'}$$

$$\frac{\Gamma \vdash e : \forall\alpha <: \tau' .\tau'' \qquad \Gamma \vdash \tau <: \tau'}{\Gamma \vdash e[\tau] : \tau''[\alpha \mapsto \tau]}$$

$$\frac{\Gamma \vdash e : \tau \qquad \tau <: \tau'}{\Gamma \vdash e : \tau'}$$

Kernel $F_{<:}$ – Subtype Relation

PLRG

We need to define **subtype relation** with type environment Γ:

$$\text{Types} \quad \tau ::= \texttt{num} \mid \tau \to \tau \mid \alpha \mid \forall \alpha <: \tau.\tau$$

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

$$\frac{}{\Gamma \vdash \tau <: \tau} \qquad \frac{\Gamma \vdash \tau_1 <: \tau_2 \qquad \Gamma \vdash \tau_2 <: \tau_3}{\Gamma \vdash \tau_1 <: \tau_3}$$

$$\frac{}{\Gamma \vdash \bot <: \tau} \qquad \frac{}{\Gamma \vdash \tau <: \top} \qquad \frac{\Gamma \vdash \tau_2 <: \tau_1 \qquad \Gamma \vdash \tau_1' <: \tau_2'}{\Gamma \vdash \tau_1 \to \tau_1' <: \tau_2 \to \tau_2'}$$

$$\frac{(\alpha <: \tau) \in \Gamma}{\Gamma \vdash \alpha <: \tau} \qquad \frac{\Gamma[\alpha <: \tau] \vdash \tau_1 <: \tau_2}{\Gamma \vdash (\forall \alpha <: \tau.\tau_1) <: (\forall \alpha <: \tau.\tau_2)}$$

Kernel $F_{<:}$ – Subtype Relation

PLRG

The **kernel** $F_{<:}$ does not allow the bound of the type variable to be different in the subtyping rule for universal types:

$$\frac{\Gamma[\alpha <: \tau] \vdash \tau_1 <: \tau_2}{\Gamma \vdash (\forall \alpha <: \tau.\tau_1) <: (\forall \alpha <: \tau.\tau_2)}$$

Kernel $F_{<:}$ – Subtype Relation

PLRG

The **kernel** $F_{<:}$ does not allow the bound of the type variable to be different in the subtyping rule for universal types:

$$\frac{\Gamma[\alpha <: \tau] \vdash \tau_1 <: \tau_2}{\Gamma \vdash (\forall \alpha <: \tau.\tau_1) <: (\forall \alpha <: \tau.\tau_2)}$$

For example, we cannot derive the following subtyping relation in kernel $F_{<:}$ because the bounds of α are different ($\texttt{num}$ and $\top$):

$$\vdash (\forall \alpha <: \top.\alpha) <: (\forall \alpha <: \texttt{num}.\alpha)$$

Full $F_{<:}$ – Subtype Relation

PLRG

We can support **full** $F_{<:}$ by allowing different bounds in the subtyping rule for universal types:

$$\frac{\Gamma \vdash \tau_2 <: \tau_1 \qquad \Gamma[\alpha <: \tau_2] \vdash \tau_1' <: \tau_2'}{\Gamma \vdash (\forall \alpha <: \tau_1 . \tau_1') <: (\forall \alpha <: \tau_2 . \tau_2')}$$

Note that the bound position (i.e., τ_1 and τ_2) is **contravariant**, while the body position (i.e., τ_1' and τ_2') is **covariant**.

Full $F_{<:}$ – Subtype Relation

PLRG

We can support **full** $F_{<:}$ by allowing different bounds in the subtyping rule for universal types:

$$\frac{\Gamma \vdash \tau_2 <: \tau_1 \qquad \Gamma[\alpha <: \tau_2] \vdash \tau_1' <: \tau_2'}{\Gamma \vdash (\forall \alpha <: \tau_1 . \tau_1') <: (\forall \alpha <: \tau_2 . \tau_2')}$$

Note that the bound position (i.e., τ_1 and τ_2) is **contravariant**, while the body position (i.e., τ_1' and τ_2') is **covariant**.

Now, we can derive the following subtyping relation in full $F_{<:}$:

$$\frac{\vdash \texttt{num} <: \top \qquad [\alpha <: \texttt{num}] \vdash \alpha <: \alpha}{\vdash (\forall \alpha <: \top . \alpha) <: (\forall \alpha <: \texttt{num} . \alpha)}$$

Full $F_{<:}$ – Subtype Relation

PLRG

We can support **full** $F_{<:}$ by allowing different bounds in the subtyping rule for universal types:

$$\frac{\Gamma \vdash \tau_2 <: \tau_1 \qquad \Gamma[\alpha <: \tau_2] \vdash \tau_1' <: \tau_2'}{\Gamma \vdash (\forall \alpha <: \tau_1.\tau_1') <: (\forall \alpha <: \tau_2.\tau_2')}$$

Note that the bound position (i.e., τ_1 and τ_2) is **contravariant**, while the body position (i.e., τ_1' and τ_2') is **covariant**.

Now, we can derive the following subtyping relation in full $F_{<:}$:

$$\frac{\vdash \texttt{num} <: \top \qquad [\alpha <: \texttt{num}] \vdash \alpha <: \alpha}{\vdash (\forall \alpha <: \top.\alpha) <: (\forall \alpha <: \texttt{num}.\alpha)}$$

However, since full $F_{<:}$ is **undecidable** but kernel $F_{<:}$ is **decidable**, kernel $F_{<:}$ is used in practice.

Next Lecture

PLRG

- Existential Types

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