

### What Is Subtyping?

A **subtype** relation is a **pre-order** on types that validates the **subsumption principle**.

- **Notation:**  $\sigma <: \tau$ .
- **Pre-order:** reflexive and transitive binary relation.
- **Subsumption:** if  $\sigma <: \tau$ , then an expression of type  $\sigma$  suffices whenever one of type  $\tau$  is required.

1

### What Is Subtyping?

The subsumption principle codifies a principle of **code re-use**.

Allows you to “re-use” a value of type  $\sigma$  in a  $\tau$  context whenever  $\sigma <: \tau$ .

Subsumption is sometimes called **inheritance**, but it is best not to confuse notions.

2

### What Is Subtyping?

To decide whether a subtype relation  $\sigma <: \tau$  is reasonable, we must check the subsumption principle.

- Every use of a value of type  $\tau$  must make sense when given a value of type  $\sigma$ .
- Must consider **all possible** uses of values of type  $\tau$ .

3

### MinML With Subtyping

We will consider the extension of MinML with subtyping.

1. Operational semantics of subtyping.
2. Type safety in the presence of subtyping.
3. Decidability of type checking.

4

### MinML With Subtyping

Two extensions:

- A subtype relation  $\sigma <: \tau$ , given by a set of subtyping rules.
- The **subsumption rule** for typing:

$$\frac{\Gamma \vdash e : \sigma \quad \sigma <: \tau}{\Gamma \vdash e : \tau}$$

**NB:** the typing relation is no longer syntax-directed!

5

### Specifying A Subtype Relation

A subtype relation is often specified in two parts:

- **Axioms** stating the fundamental forms of subtyping.
- **Variance Principles** stating how type constructors interact with subtyping.

6

### Specifying A Subtype Relation

The complete subtype relation is taken to be the least pre-order ...

- containing the axioms, and ...
- closed under the variance principles.

To ensure that it is a pre-order we implicitly include these rules:

$$\frac{}{\tau <: \tau} \quad \frac{\rho <: \sigma \quad \sigma <: \tau}{\rho <: \tau}$$

7

### Arithmetic Subtyping

Arithmetic subtyping axiom:

$$\overline{\text{int} <: \text{float}}$$

Motivated by the inclusion  $\mathbb{Z} \subseteq \mathbb{R}$ .

(There are no variance rules for atomic types.)

8

### Safety of Arithmetic Subtyping

Assume we have floating point constants  $f$ .

Assume we have (at least) floating point addition:

$$\frac{\Gamma \vdash e_1 : \text{float} \quad \Gamma \vdash e_2 : \text{float}}{\Gamma \vdash e_1 + e_2 : \text{float}}$$

Is the resulting language type safe?

9

### Arithmetic Subtyping

#### Theorem 1 (Preservation)

If  $e : \tau$  and  $e \mapsto e'$ , then  $e' : \tau$ .

The proof is exactly as before, because it proceeds by **induction on evaluation**.

Just add new cases for floating point arithmetic, e.g.,  $f_1 + f_2 \mapsto f$ , where  $f = f_1 \oplus f_2$ .

10

### Arithmetic Subtyping

#### Theorem 2 (Progress)

If  $e : \tau$ , then either  $e$  value or  $e \mapsto e'$  for some  $e'$ .

The proof is by **induction on typing**, so a **new case** must be considered!

Suppose that  $e : \tau$  because  $e : \tau'$  and  $\tau' <: \tau$ .

By induction either  $e$  value or there exists  $e'$  such that  $e \mapsto e'$ .

11

### Arithmetic Subtyping

Suppose that  $e_1 + e_2 : \text{float}$  because  $e_1 : \text{float}$  and  $e_2 : \text{float}$ .

Suppose further that  $e_1$  value and  $e_2$  value. Is there a value  $v$  such that  $e_1 + e_2 \mapsto v$ ?

**This is not obvious!**

12

### Arithmetic Subtyping

What are the values of type `float`?

**Lemma 3 (Canonical Forms)**

If  $e$  value and  $e : \text{float}$ , then either

1.  $e$  is a **floating point constant**  $f$ , or
2.  $e$  is an **integer constant**  $n$  (because of subsumption).

So we must define  $v_1 + v_2$  when either  $v_1$  or  $v_2$  is an **integer constant!**

Otherwise progress fails.

13

### Tuple Subtyping

Tuple subtyping axiom:

$$\frac{(m > n)}{\tau_1 * \dots * \tau_m <: \tau_1 * \dots * \tau_n}$$

The **wider** tuple is a subtype of the **narrower!**

- Projections from a narrow tuple apply also to a wide tuple.
- Conversely, a 9-tuple has no 10th component.

14

### Record Subtyping

Record subtyping axiom:

$$\frac{(m > n)}{\{l_1 : \tau_1, \dots, l_m : \tau_m\} <: \{l_1 : \tau_1, \dots, l_n : \tau_n\}}$$

Meaning depends on whether record types are **ordered** (C-like) or **unordered** (ML-like):

- **Ordered:** can drop fields **at the end** of a record.
- **Unordered:** can drop fields **anywhere** within a record.

15

### Record Subtyping

Evaluation of record projection:

$$\{l_1 : v_1, \dots, l_n : v_n\}.l_i \mapsto v_i$$

But how do we find the field labelled  $l_i$ ?

- Without subtyping we can use the type of the record to predict the position of any field.
- With subtyping the type does not reveal the shape of the record; there may be many more fields than the type specifies!

16

### Sum Subtyping

What is the appropriate subtyping axiom for  $n$ -ary sums?

$$\frac{(m < n)}{\tau_1 + \dots + \tau_m <: \tau_1 + \dots + \tau_n}$$

The **smaller** sum is the subtype. Why?

- An element  $\text{in}_i(v_i)$  of the smaller is also an element of the larger.
- Case analysis on the supertype covers the subtype.

17

## Variance Principles

A **variance** principle tells how a type constructor interacts with subtyping in each position.

- **Co-variance**: the constructor **preserves** subtyping.
- **Contra-variance**: the constructor **reflects** subtyping.
- **Invariance**: the constructor **precludes** subtyping.

18

## Tuple Variance

Without further specification, tuples are **invariant**.

For example,

`int*float`  $\not\prec$  `float*float`

even if `int`  $\prec$  `float`!

19

## Tuple Variance

To allow these subtyping relationships, specify that tuples are **covariant**:

$$\frac{\forall 1 \leq i \leq n \ \sigma_i \prec \tau_i}{\sigma_1 * \dots * \sigma_n \prec \tau_1 * \dots * \tau_n}$$

Subtyping is **preserved** in each field of the tuple.

20

## Tuple Variance

Why is covariance safe?

- Must check that it validates subsumption.
- What can we do with  $\tau_1 * \dots * \tau_n$ ?
  - Extract  $i$ th component and use it as a value of type  $\tau_i$ .
- If we actually have a  $\sigma_1 * \dots * \sigma_n$ , the  $i$ th component is a  $\sigma_i$ .
  - But we can use it as a  $\tau_i$ !

21

## Sum Variance

Sums are also co-variant:

$$\frac{\forall 1 \leq i \leq n \ \sigma_i \prec \tau_i}{\sigma_1 + \dots + \sigma_n \prec \tau_1 + \dots + \tau_n}$$

Check: case analysis on the supertype.

- The  $i$ th case expects a value of type  $\tau_i$ ;
- By subsumption it is OK to provide a value of type  $\sigma_i$ .

22

## Function Subtyping

What variance principles should apply to  $\tau_1 \rightarrow \tau_2$ ?

- When is it sensible to have

$$\sigma_1 \rightarrow \sigma_2 \prec \tau_1 \rightarrow \tau_2?$$

- What can we do with a value of the supertype? Is a value of the subtype acceptable?

23

### Function Variance

Suppose that  $\text{int} <: \text{float}$ . Which should hold? Which should not? Why?

- $\text{int} \rightarrow \text{int} <: \text{float} \rightarrow \text{float}$ ?
- $\text{float} \rightarrow \text{float} <: \text{int} \rightarrow \text{int}$ ?
- $\text{float} \rightarrow \text{int} <: \text{int} \rightarrow \text{float}$ ?
- $\text{int} \rightarrow \text{float} <: \text{float} \rightarrow \text{int}$ ?

24

### Function Variance

What can we do with a value of type  $\tau_1 \rightarrow \tau_2$ ?

- Apply it to an argument of type  $\tau_1$ .
- Use the result as a value of type  $\tau_2$ .

25

### Function Variance

Suppose now that  $f : \sigma_1 \rightarrow \sigma_2$ .

- When does it make sense to apply it to a value of type  $\tau_1$ ?  
**Only if**  $\tau_1 <: \sigma_1$ !
- When does it make sense to use its result as a value of type  $\tau_2$ ?  
**Only if**  $\sigma_2 <: \tau_2$ !

26

### Function Variance

The function type constructor is

- **Covariant** in the **range**.
- **Contravariant** in the **domain**.

The variance rule for functions is

$$\frac{\tau_1 <: \sigma_1 \quad \sigma_2 <: \tau_2}{\sigma_1 \rightarrow \sigma_2 <: \tau_1 \rightarrow \tau_2}$$

27

### Reference Variance

What is the appropriate variance principle for reference types?

- $\text{ref}(\sigma) <: \text{ref}(\tau)$  if ... ?

How can a value of type  $\text{ref}(\tau)$  be used?

- Fetch its contents, use as a value of type  $\tau$ .
- Replace its contents with a value of type  $\tau$ .

28

### Reference Variance

If  $r$  has type  $\text{ref}(\sigma)$ , then its contents are of type  $\sigma$ .

- Contents have type  $\tau$  only if  $\sigma <: \tau$ .
- **Covariance!**

Suppose we wish to **store** a value of type  $\tau$  in  $r$ .

- Valid only if  $\tau <: \sigma$ .
- **Contravariance!**

29

### Reference Variance

Suppose that  $r$  has type `ref(int)`.

If reference types were covariant,

- Then  $r$  would have type `ref(float)` too.
- So we could store  $\pi$  into  $r$ .
- But the contents of  $r$  must be an integer!

30

### Reference Variance

Suppose that  $r$  has type `ref(float)`.

If reference types were contravariant,

- Then  $r$  would have type `ref(int)` too.
- Storing an integer is OK, since every integer is a float.
- But the contents of  $r$  might be  $\pi$ !

31

### Reference Variance

Either way we lose type safety.

Conclusion: reference types are **invariant**!

- Type of a reference cell does not change by subtyping!

Similar conclusions apply to

- **Mutable record** types whose fields are assignable (*e.g.*, `struct`'s).
- **Mutable arrays** whose elements may be assigned.

32

### Summary

Subtyping supports code reuse by **subsumption**.

Choosing subtyping principles is tricky.

- Unsoundness.
- Potential for inefficiency.

33