

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 Principle:** if $\sigma <: \tau$, then an expression of type σ suffices whenever one of type τ is required.

What Is Subtyping?

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

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

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

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 τ must make sense when given a value of type σ .
- Must consider **all possible** uses of values of type τ .

That is, need to determine whether every **introductory form** of the subtype can be safely manipulated by every **eliminary form** of the supertype.

Subsumption

To ensure that subtyping is a pre-order we insist that the following structural rules be admissible:

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

Either tacitly include these rules as primitive or prove they are admissible for a given set of subtyping rules

The subsumption rule is the fundamental definition of subtyping:

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

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

Numeric Subtyping

Good way to get subtyping wrong is to equate it with subsetting of values.

For example, it's tempting to postulate the subtyping relationship:

$$\text{int} <: \text{rat} <: \text{real}$$

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

But this is unrealistic due to nature of floating point representation!

Product Subtyping

One form of subtyping for tuples, called **width subtyping**, is specified by the subtyping rule:

$$\frac{J \subseteq I}{\prod_{i \in I} \tau_i <: \prod_{j \in J} \tau_j}$$

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.

6

Product Subtyping

In the common case where the index sets are initial segments of the natural numbers, this specializes to:

$$\frac{m \geq n}{\langle \tau_1, \dots, \tau_m \rangle <: \langle \tau_1, \dots, \tau_n \rangle}$$

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.

7

Record Subtyping

Width subtyping rule for records, where the index sets are finite sets of symbols, is:

$$\frac{m \geq n}{\langle l_1 : \tau_1, \dots, l_m : \tau_m \rangle <: \langle l_1 : \tau_1, \dots, l_n : \tau_n \rangle}$$

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.

8

Record Subtyping

Evaluation of record projection:

$$\langle l_1 : v_1, \dots, l_n : v_n \rangle \cdot l_i \mapsto v_i$$

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! Options include mapping (e.g., hash) functions and coercion (immutable records only).

9

Sum Subtyping

What is the appropriate width subtyping rule for finite sums?

$$\frac{I \subseteq J}{\sum_{i \in I} \tau_i <: \sum_{j \in J} \tau_j}$$

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

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

10

Sum Subtyping

In the common case where the index sets are initial segments of the natural numbers, this specializes to:

$$\frac{m \leq n}{[\tau_1 : \tau_1, \dots, \tau_m : \tau_m] <: [\tau_1 : \tau_1, \dots, \tau_n : \tau_n]}$$

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

11

Variance Principles

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

- **Covariance**: the constructor **preserves** subtyping.
- **Contravariance**: the constructor **reverses** subtyping.
- **Invariance**: the constructor **precludes** subtyping.

12

Product Subtyping

Subtyping rule for tuple types specifies that tuples are **covariant**:

$$\frac{(\forall i \in I) \sigma_i <: \tau_i}{\prod_{i \in I} \sigma_i <: \prod_{i \in I} \tau_i}$$

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

Called **depth subtyping** since it applies subtyping **within** components.

13

Product Subtyping

For n-tuples the rule specializes to:

$$\frac{\sigma_1 <: \tau_1 \quad \dots \quad \sigma_n <: \tau_n}{\langle \sigma_1, \dots, \sigma_n \rangle <: \langle \tau_1, \dots, \tau_n \rangle}$$

14

Record Subtyping

Covariant depth subtyping also applies to record types.:

$$\frac{\sigma_1 <: \tau_1 \quad \dots \quad \sigma_n <: \tau_n}{\langle l_1 : \sigma_1, \dots, l_n : \sigma_n \rangle <: \langle l_1 : \tau_1, \dots, l_n : \tau_n \rangle}$$

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

15

Product Variance

Why is covariance safe?

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

16

Sum Subtyping

Depth subtyping for sums is also based on covariance:

$$\frac{(\forall i \in I) \sigma_i <: \tau_i}{\sum_{i \in I} \sigma_i <: \sum_{i \in I} \tau_i}$$

Check: case analysis on the supertype.

- The i th case expects a value of type τ_i .
- By subsumption it is OK to provide a value of type σ_i .

17

Sum Subtyping

When specialized to symbolic labels as index sets, the covariance principle for sum types takes the following form:

$$\frac{\sigma_1 <: \tau_1 \quad \dots \quad \sigma_n <: \tau_n}{[\iota_1 : \sigma_1, \dots, \iota_n : \sigma_n] <: [\iota_1 : \tau_1, \dots, \iota_n : \tau_n]}$$

18

Function Subtyping

What variance principles should apply to $\sigma \rightarrow \tau$?

- When is it sensible to have

$$\sigma \rightarrow \tau <: \sigma' \rightarrow \tau'?$$

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

19

Function Variance

What can we do with a value of type $\sigma' \rightarrow \tau'$?

- Apply it to an argument of type σ' .
- Use the result as a value of type τ' .

20

Function Variance

Suppose now that $f : \sigma \rightarrow \tau$.

- When does it make sense to apply it to a value of type σ' ? **Only if** $\sigma' <: \sigma$!
- When does it make sense to use its result as a value of type τ' ? **Only if** $\tau <: \tau'$!

21

Function Variance

The function type constructor is

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

The variance rule for functions is

$$\frac{\sigma' <: \sigma \quad \tau <: \tau'}{\sigma \rightarrow \tau <: \sigma' \rightarrow \tau'}$$

22

Safety for Subtyping

Proving safety in the presence of subtyping is a bit delicate.

Subsumption means that static type only reveals partial information regarding underlying values.

So proof of preservation and progress, and statements and proofs of underlying lemmas, change accordingly.

Illustrated here by considering safety of product types in isolation.

23

Tuple Types: Static Semantics

$$\frac{(\forall i \in I) \Gamma \vdash e_i : \tau_i}{\Gamma \vdash \langle e_i \rangle_{i \in I} : \prod_{i \in I} \tau_i}$$

$$\frac{\Gamma \vdash e : \prod_{i \in I} \tau_i \quad j \in I}{\Gamma \vdash e \cdot j : \tau_j}$$

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

$$\frac{m \geq n}{\langle \tau_1, \dots, \tau_m \rangle <: \langle \tau_1, \dots, \tau_n \rangle}$$

$$\frac{\sigma_1 <: \tau_1 \quad \dots \quad \sigma_n <: \tau_n}{\langle \sigma_1, \dots, \sigma_n \rangle <: \langle \tau_1, \dots, \tau_n \rangle}$$

24

Safety for Tuples

Lemma 1 (Structurality)

1. The tuple subtyping relation is reflexive and transitive.
2. The typing judgement $\Gamma \vdash e : \tau$ is closed under weakening and substitution.

25

Safety for Tuples

Lemma 2 (Inversion)

1. If $e \cdot j : \tau$ then $e : \prod_{i \in I} \tau_i$, $j \in I$, and $\tau_j <: \tau$.
2. If $\langle e_i \rangle_{i \in I} : \tau$ then $\prod_{i \in I} \sigma_i <: \tau$ where $e_i : \sigma_i$ for each $i \in I$.
3. If $\sigma <: \prod_{j \in J} \tau_j$, then $\sigma = \prod_{i \in I} \sigma_i$ for some I and some types σ_i for $i \in I$.
4. If $\prod_{i \in I} \sigma_i <: \prod_{j \in J} \tau_j$, then $J \subseteq I$ and $\sigma_j <: \tau_j$ for each $j \in J$.

Proof: By induction on the typing rules, taking special care with the subsumption rule. ■

26

Tuple Types: Dynamic Semantics

$$\frac{\{(\forall i \in I) e_i \text{ val}\}}{\langle e_1, \dots, e_i \rangle \text{ val}}$$

$$\left\{ \frac{e_j \mapsto e'_j \quad (\forall i \neq j) e'_i = e_i}{\langle e_1, \dots, e_i \rangle \mapsto \langle e'_1, \dots, e'_i \rangle} \right\}$$

$$\frac{e \mapsto e'}{e \cdot j \mapsto e' \cdot j}$$

$$\frac{\langle e_1, \dots, e_i \rangle \text{ val}}{\langle e_1, \dots, e_i \rangle \cdot j \mapsto e_j}$$

Bracketed rule omitted for lazy semantics and included for eager semantics.

27

Safety for Tuples

Theorem 3 (Preservation)

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

Proof: By induction on dynamic semantics. ■

28

Safety for Tuples

Theorem 4 (Canonical Forms)

If $e \text{ val}$ and $e : \prod_{j \in J} \tau_j$, then e is of the form $\langle e_i \rangle_{i \in I}$ where $J \subseteq I$ and $e_j : \tau_j$ for each $j \in J$.

Proof: By induction on static semantics, taking account of the definition of values. Note that the value of a tuple type is, in general, larger than is predicted by its type. ■

29

Safety for Records

Theorem 5 (Progress)

If $e : \tau$, then either e val, or there exists e' such that $e \mapsto e'$.

Proof: By induction on static semantics. ■

30

Summary

Subtyping supports code reuse by **subsumption**.

Choosing subtyping principles is tricky.

- Unsoundness.
- Potential for inefficiency.

31