Subtyping: Definition

- A subtyping relation is a preorder $\leq$ between types used by the *subsumption* rule:
  \[
  \frac{\Gamma \vdash e : t_1 \quad t_1 \leq t_2}{\Gamma \vdash e : t_2}
  \]

- If $t_1 \leq t_2$ then we say that $t_1$ is a *subtype* of $t_2$.

NB: A preorder is a relation that is reflexive and transitive (but not necessarily antisymmetric)

Interpretations of Subtyping

If $t_1 \leq t_2$ then...

1. The type $t_1$ is more precise (less general) description of a value than $t_2$.
2. Either every value of type $t_1$ also has type $t_2$, or, there is a standard way to convert values of type $t_1$ to values of type $t_2$.
3. In any context where a value of type $t_2$ is expected, it is acceptable to provide a value of type $t_1$.

Examples

- $\text{Integer} \leq \text{Number} \leq \text{Object}$
- $\text{char} \leq \text{int} \leq \text{long} \leq \text{float} \leq \text{double}$
- $\text{even} \leq \text{nat} \quad \text{odd} \leq \text{nat}$
Subtyping is not Inheritance!

- These concepts are conflated in C++, Java
  - Subclasses always generate subtypes

- But, these are really orthogonal concepts
  - Can have subtyping without inheritance
    - e.g., primitive types in C
  - Can have inheritance without subtyping
    - e.g., C++ private inheritance

Example Typing Derivation

- Assume \( \text{int} \subseteq \text{real} \)
- Then

  \[
  \begin{align*}
  3 : \text{int} & \quad \text{int} \subseteq \text{real} \\
  3 : \text{real} & \quad 2.5 : \text{real} \\
  (3, 2.5) : \text{real} \times \text{real}
  \end{align*}
  \]

Language Design

- Is the choice of subtyping arbitrary?
  - Given the dynamic semantics, only certain choices for subtyping avoid run-time type errors.
    - Asking for trouble when this is ignored.
  - However, a language need not include all "natural" subtyping relationships.
    - Implementation costs
    - Methodological/simplicity arguments
  - Structural vs. by-name subtyping

Inclusive Viewpoint

- Suppose we just throw in the subsumption rule into the type system we saw last week.
  - With no change to operational semantics
  - No run-time data coercions.
- What definitions of \( \subseteq \) are sound?
- Informal methodology for deciding \( t_1 \subseteq t_2 \):
  - What can you do with values of type \( t_2 \)?
  - Question: would it be safe to apply these operations to an arbitrary value of type \( t_1 \)?
Pair Types

- Suppose even ≤ nat.
  - Which of the following are ok?

  1. even*string ≤ nat*string
  2. nat*string ≤ even*string
  3. even*even ≤ nat*nat

\[
t_1 \cdot t_2 \leq t_1' \cdot t_2'
\]

Tuple Types

- Suppose even ≤ nat.
  - Which of the following are ok?

  1. even*even*even ≤ nat*nat*nat
  2. even*string*nat ≤ even*string
  3. even*string ≤ even*string*nat
  4. even*even*even ≤ nat*nat

\[
t_1 \cdot \ldots \cdot t_{n+m} \leq t_1' \cdot \ldots \cdot t_n'
\]

Function Types

- Suppose even ≤ nat.
  - Which of the following are ok?

  1. even -> even ≤ even -> nat
  2. even -> nat ≤ even -> even
  3. even -> even ≤ nat -> even
  4. nat -> even ≤ even -> even
  5. even -> even ≤ nat -> nat

\[
t_1 \rightarrow t_2 \leq t_1' \rightarrow t_2'
\]

Reference Types

- Suppose even ≤ nat.
  - Which of the following are ok?

  1. even ref ≤ nat ref
  2. nat ref ≤ even ref

\[
t_1 \text{ ref} \leq t_2 \text{ ref}
\]
Vector and Array Types

- Vector (immutable array)
  - Supports subscript operation
- Array
  - Supports subscript and update operations
- Which are ok?
  1. even vector \(\leq\) nat vector
  2. even array \(\leq\) nat array

Java Arrays

- The Java language is defined so that
  \(\text{Integer[]} \leq \text{Object[]}\)
- We've just argued that this is "unsafe"
- How does Java get around this problem?

Coercive Viewpoint

- \(t_1\) is a subtype of \(t_2\) when...
  - there is a standard way to convert values of type \(t_1\) to values of type \(t_2\).
  - Compiler will automatically insert run-time coercions where required
  - Coercions may involve actual work.
- Canonical example: \(\text{int} \leq \text{float}\)
  - Other coercions? \(\text{float} \rightarrow \text{int} \rightarrow \text{float}\)

Coherence

- Idea:
  - the way the compiler can insert implicit coercions shouldn't change the meaning of a program
  - Frequently an issue when subtyping is combined with overloading
    \((6 / 7) \times 7.0\)
  - Even when there are fixed rules for inserting coercions, don't want surprising behavior
    \((1 / 3) + 15 \approx 5.3333\) ???