Subtyping

February 27, 2002

CS 131: Programming Languages

Subtyping: Definition

- A *subtyping* relation is a preorder $\preccurlyeq$ between types used by the *subsumption* rule:

$$\frac{tenv \vdash exp : type_1 \qquad type_1 \preccurlyeq type_2}{tenv \vdash exp : type_2}$$

- If $t_1 \preccurlyeq t_2$ 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 $t_1 \preccurlyeq t_2$ 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

$$\texttt{Integer} \preccurlyeq \texttt{Number} \preccurlyeq \texttt{Object}$$

$$\texttt{char} \preccurlyeq \texttt{int} \preccurlyeq \texttt{long} \preccurlyeq \texttt{float} \preccurlyeq \texttt{double}$$

$$\texttt{even} \preccurlyeq \texttt{nat} \qquad \texttt{odd} \preccurlyeq \texttt{nat}$$

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

$$\texttt{int} \leqslant \texttt{real}$$

- Then

$$\frac{\dfrac{}{\texttt{3} : \texttt{int}} \quad \dfrac{}{\texttt{int} \leqslant \texttt{real}}}{\texttt{3} : \texttt{real}} \qquad \frac{}{\texttt{2.5} : \texttt{real}}$$

$$\texttt{(3,2.5)} : \texttt{real*real}$$

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 class.
  - With no change to operational semantics
  - No run-time data coercions.
- What definitions of $\leqslant$ are sound?
- Informal methodology for deciding $t_1 \leqslant t_2$:
  - 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** $\preccurlyeq$ **nat**.
  - Which of the following are ok?

1. **even*string** $\preccurlyeq$ **nat*string**
2. **nat*string** $\preccurlyeq$ **even*string**
3. **even*even** $\preccurlyeq$ **nat*nat**

$$\frac{}{type_1 * type_2 \preccurlyeq type'_1 * type'_2}$$

Tuple Types

- Suppose **even** $\preccurlyeq$ **nat**.
  - Which of the following are ok?

1. **even*even*even** $\preccurlyeq$ **nat*nat*nat**
2. **even*string*nat** $\preccurlyeq$ **even*string**
3. **even*string** $\preccurlyeq$ **even*string*nat**
4. **even*even*even** $\preccurlyeq$ **nat*nat**

Function Types

- Suppose even $\preccurlyeq$ nat.
- Which of the following are ok?

1. **even -> even** $\preccurlyeq$ **even -> nat**
2. **even -> nat** $\preccurlyeq$ **even -> even**
3. **even -> even** $\preccurlyeq$ **nat -> even**
4. **nat -> even** $\preccurlyeq$ **even -> even**
5. **even -> even** $\preccurlyeq$ **nat -> nat**

$$\frac{}{type_1 \texttt{->} type_2 \preccurlyeq type'_1 \texttt{->} type'_2}$$

Reference Types

- Suppose **even** $\preccurlyeq$ **nat**.
  - Which of the following are ok?

1. **even ref** $\preccurlyeq$ **nat ref**
2. **nat ref** $\preccurlyeq$ **even ref**

$$\frac{}{type_1 \ \texttt{ref} \preccurlyeq type_2 \ \texttt{ref}}$$

Vector and Array Types

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

Java Arrays

- The Java language is defined so that
  Integer[] $\preccurlyeq$ **Object[]**
- We've just argued that this is "unsafe"
- How does Java get around this problem?

Coercive Viewpoint

- $type_1$ is a subtype of $type_2$ when...
  - there is a standard way to convert values of type $type_1$ to values of type $type_2$.
  - Compiler will automatically insert run-time coercions where required
  - Coercions may involve actual work.
- Canonical example: **int** $\preccurlyeq$ **float**
  - Other coercions?
    float->int to **int->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) * 7.0`
  - Even when there are fixed rules for inserting coercions, don't want surprising behavior
    `(1 / 3) + 15 == 5.33333 (!??)`