Subtyping

October 25, 2001

CS 131: Programming Languages

Subtyping: Definition

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

$$\frac{\Gamma \vdash e : t_1 \qquad t_1 \preccurlyeq t_2}{\Gamma \vdash e : t_2}$$

- 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

$$\texttt{int} \preccurlyeq \texttt{real}$$

- Then

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

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

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 week.
  - With no change to operational semantics
  - No run-time data coercions.
- What definitions of $\preccurlyeq$ are sound?
- Informal methodology for deciding $\texttt{t}_1 \preccurlyeq \texttt{t}_2$:
  - What can you do with values of type $\texttt{t}_2$?
  - Question: would it be safe to apply these operations to an arbitrary value of type $\texttt{t}_1$?

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

$$\frac{}{t_1 * t_2 \preccurlyeq t_1' * t_2'}$$

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

$$\frac{}{t_1 * \ldots * t_{n+m} \preccurlyeq t_1' * \ldots * t_n'}$$

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

$$\frac{}{t_1 \to t_2 \preccurlyeq t_1' \to t_2'}$$

Reference Types

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

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

$$\frac{}{t_1 \text{ ref} \preccurlyeq t_2 \text{ ref}}$$

Vector and Array Types

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

- 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: `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 ???`