Computer Science 131
Programming Languages

November 2, 2000
Subtyping
Preorders

• A preorder is a relation that is
  - Reflexive
  - Transitive
  - Needs not be antisymmetric
Subtyping: Definition

• A subtyping relation is a preorder $\preceq$ between types validating the subsumption rule:

\[
\Gamma \vdash e : t_1 \quad t_1 \preceq t_2 \\
\therefore \quad \Gamma \vdash e : t_2
\]

• If $t_1 \preceq t_2$ then we say that $t_1$ is a subtype of $t_2$. 
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. Every value of type $t_1$ also has type $t_2$.
  3. There is a standard way to convert values of type $t_1$ to values of type $t_2$.
  4. In any context where a value of type $t_2$ is expected, it is acceptable to provide a value of type $t_1$. 
Examples

```
Integer ⊑ Number ⊑ Object

char ⊑ int ⊑ long ⊑ float ⊑ double

even ⊑ nat       odd ⊑ nat
```
Subtyping is not Inheritance!

• These concepts are conflated in C++, Java
  - Subclasses always generate subtypes
• But, these are really orthogonal concepts
  - Could have subtyping without inheritance
  - Could have inheritance without subtyping
Example Typing Derivation

• Assume

\[ \text{int} \preceq \text{real} \]

• Then

\[
\begin{array}{ccc}
3 : \text{int} & \text{int} \preceq \text{real} & 2.5 : \text{real} \\
\hline
3 : \text{real} & 2.5 : \text{real} & (3,2.5) : \text{real} * \text{real}
\end{array}
\]
Language Design

• Is the choice of subtyping arbitrary?
  - Given the operational semantics, only certain choices for subtyping are sound.
    • Asking for trouble when this is ignored.
  - However, a language 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 a NQSML-like type system.
  - With no change to operational semantics
  - No run-time data coercions.
- What definitions of $\subseteq$ are sound?
- Informal methodology: can $\mathfrak{t}_1 \subseteq \mathfrak{t}_2$
  - What can you do with values of type $\mathfrak{t}_2$?
  - Question: would it be safe to apply these operations to an arbitrary value of type $\mathfrak{t}_1$?
Pair Types

- Suppose `even \leq \text{nat}`.
- Which of the following are ok?

  1. `even*string \leq \text{nat}*string`
  2. `\text{nat}*string \leq \text{even}*string`
  3. `even*even \leq \text{nat}*\text{nat}`
Pair Types

• In general,

\[
\frac{t_1 \preceq t_1' \quad t_2 \preceq t_2'}{t_1 \ast t_2 \preceq t_1' \ast t_2'}
\]
Tuple Types

• Suppose $\text{even} \preceq \text{nat}$.
• Which of the following are ok?

1. $\text{even*even*even} \preceq \text{nat*nat*nat}$
2. $\text{even*string*nat} \preceq \text{even*string}$
3. $\text{even*string} \preceq \text{even*string*nat}$
4. $\text{even*even*even} \preceq \text{nat*nat}$
Tuple Types

• It follows that,

\[
\begin{align*}
  t_1 \preceq t_1' & \quad \ldots \quad t_n \preceq t_n' \\
  t_1 \ast \ldots \ast t_{n+m} & \preceq t_1' \ast \ldots \ast t_n'
\end{align*}
\]
Function Types

• Suppose $\text{even} \preceq \text{nat}$.
• Which of the following are ok?

1. $\text{even} \rightarrow \text{even} \preceq \text{even} \rightarrow \text{nat}$
2. $\text{even} \rightarrow \text{nat} \preceq \text{even} \rightarrow \text{even}$
3. $\text{even} \rightarrow \text{even} \preceq \text{nat} \rightarrow \text{even}$
4. $\text{nat} \rightarrow \text{even} \preceq \text{even} \rightarrow \text{even}$
5. $\text{even} \rightarrow \text{even} \preceq \text{nat} \rightarrow \text{nat}$
Function Types

• In general,

\[
\begin{align*}
    t_1' & \leq t_1 \\
    t_2 & \leq t_2' \\
\hline
    t_1 \rightarrow t_2 & \leq t_1' \rightarrow t_2'
\end{align*}
\]
Reference Types

• Suppose \( \text{even} \sqsubseteq \text{nat} \).

• Which of the following are ok?

1. \( \text{even} \text{ Ref} \sqsubseteq \text{nat Ref} \)
2. \( \text{nat Ref} \sqsubseteq \text{even Ref} \)
Reference Types

• In general,

\[
\frac{t_1 = t_2}{t_1 \text{ Ref} \preceq t_2 \text{ Ref}}
\]
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
  \[ \text{Integer[]} \subseteq \text{Object[]} \]
• We've just argued that this is "unsafe"
• How does Java get around this problem?
Coercive Viewpoint

• $t_1$ 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: $\text{int} \leq \text{float}$
Coercive Viewpoint

• Suppose we have a coercion function
  \( c : \text{int} \rightarrow \text{float} \)

• What other natural coercions can we define?
  \( \text{int}\times\text{int} \rightarrow \text{float}\times\text{float} \)
  \( \text{float}\times\text{float} \rightarrow \text{int}\times\text{int} \)
  \( \text{int}\times\text{float} \rightarrow \text{int}\times\text{int}\times\text{int} \)
  \( (\text{float} \rightarrow \text{int}) \rightarrow (\text{int} \rightarrow \text{float}) \)
  \( (\text{int} \rightarrow \text{int}) \rightarrow (\text{float} \rightarrow \text{float}) \)
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
    \[(6 / 7) * 7.0\]
  - Even when there are fixed rules for inserting coercions, don't want surprising behavior
    \[(1 / 3) + 15\]
Information Loss

• Suppose \texttt{Integer \leq Numeric}, and we want a function that takes a numeric object and adds it to itself.

• So far, the best we can do is write
  \texttt{double : Numeric -> Numeric}

• But this loses information. If
  \texttt{n : Integer}

then

  \texttt{double(n) : Numeric}. 
Can Polymorphism Help?

• If we could say

\[
\text{double} : \forall \alpha. \alpha \rightarrow \alpha
\]

then

\[
\text{double[Integer]}(n) : \text{Integer}
\]

But we can't pass an arbitrary object to \text{double} because the code requires the argument have a method for addition.
Bounded Polymorphism

• Extension:
  - Polymorphic functions that take not an arbitrary type, but any subtype of a given type.
    
    $\text{double} : \forall \alpha \leq \text{Numeric} . \alpha \rightarrow \alpha$

• Then

  $\text{double}[\text{Integer}](n) : \text{Integer}.$