SUBTYPING EXTRAVAGANZA

What is subtyping?

- subtyping = subsets
- subtyping = subclasses
- subtyping = substitutability
- subtyping = polymorphism

All valid perspectives

but not all the same meaning

Subtyping Semantically = implicit coercions

types

$A <: B \implies \mathrm{coerce}_{A,B} : [\![A]\!] \to [\![B]\!]$

subtype relation

the semantics of a type

function converting values of $[\![A]\!]$ to values of $[\![B]\!]$

How can we take advantage of this?

Some languages let programmers define their own coercions to extend the subtyping relation.

C++ Scala

What can go wrong?

Ambiguity

In C++ and Scala, how the compiler inserts coercions can affect the semantics of a program.

Preventing Ambiguity

$A <: B$, $B <: C$, $A <: C$

holds due to transitivity

$\implies \mathrm{coerce}_{A,B} \,;\, \mathrm{coerce}_{B,C} = \mathrm{coerce}_{A,C}$

sequential composition

$A <: A \implies \mathrm{coerce}_{A,A} = \mathrm{identity}_A$

holds due to reflexivity

Number Conversion Example

int <: double — coercion is obvious

double <: int — use floor to coerce

Convenient, but what goes wrong?

$$\begin{tikzcd} \underset{\text{double}}{2.5} \arrow[r, "\text{coerce}"] \arrow[rr, bend right, "\text{coerce}"'] & \underset{\text{int}}{2} \arrow[r, "\text{coerce}"] & \underset{\text{double}}{2.0} \end{tikzcd}$$

≠ identity!

Messes up transformations like inlining

Subtyping Categorically

But first!

What's a category???

A Category is:

1. A collection of objects, e.g. A, B, C
2. For each pair of objects A and B, a collection of morphisms, e.g.

   $f: A \to B$ or $A \xrightarrow{f} B$

   morphism (f), domain (A), codomain (B)

So far this looks like a graph but there's more!

A Category also has:

3. For every object A, a "special" morphism $id_A : A \to A$ (called the identity)
4. For every $A \xrightarrow{f} B \xrightarrow{g} C$, a morphism $A \xrightarrow[g \circ f]{f;g} C$ (called composition)

and more to come later...

Example: Category of Sets (called Set)

1. The objects are all possible sets
2. The morphisms from set A to set B are all possible functions from A to B
3. id_A is the identity function: $\lambda a.\, a$
4. morphism composition is function composition

Classic example, but not all morphisms are functions!

Example: Subtyping (for some given language) as a category

1. The objects are the types
2. There exists a unique morphism from A to B iff A is a subtype of B
3. id_A exists because of reflexivity
4. composition is defined because of transitivity

Thin categories (i.e. ones with at most one morphism between any two objects) correspond to Preorders (i.e. reflexive transitive binary relations)

A Category (lastly) also has:

5. For all $A \xrightarrow{f} B$, $id_A ; f = f = f ; id_B$

   i.e. composition with identities does nothing

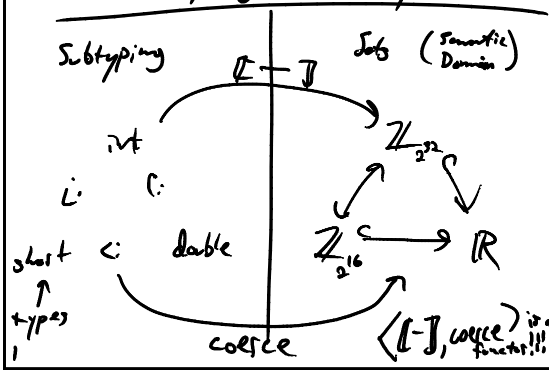
6. For all $A \xrightarrow{f} B \xrightarrow{g} C \xrightarrow{h} D$,

   $(f;g);h = f;(g;h)$

   i.e. composition is associative

Thus, given a (possibly empty) path of morphisms from A to B, there is an unambiguous way to compose that path into a morphism from A to B!

Subtyping Categorically



A Functor F from C to D is: ← categories

1. A function$_F$ from objects of C to objects of D

2. A Function from $A \xrightarrow{f} B$ to $F(A) \xrightarrow{F(f)} F(B)$

plus more!

A Functor also has:

3. $F(id_A) = id_{F(A)}$

   (preserves identities)

4. $F(f;g) = F(f);F(g)$

   (preserves composition)

In particular, for subtyping

3 ⇒ $\text{coerce}_{A,A} = id_{[[A]]}$

4 ⇒ $\text{coerce}_{A,B} ; \text{coerce}_{B,C} = \text{coerce}_{A,C}$

so unambiguity is related to categorical structure

Sneak Peek:

What does

"Hello" + 1+2

evaluate to?