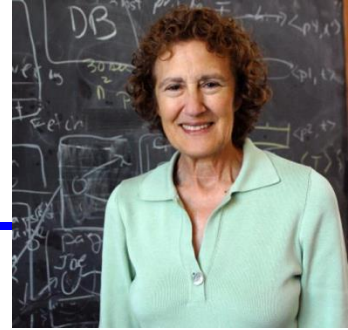# CMSC 330: Organization of Programming Languages

Subtyping

Subtyping

- The Liskov Substitution Principle:
  - Let `P(x)` be a property provable about objects x of type T. Then `P(y)` should be true for objects y of type S where S is a subtype of T.
- In other words
  - If S is a subtype of T, then an S can be used anywhere a T is expected
- Commonly used in object-oriented programming
  - Subclasses can be used where superclasses expected .
  - This is a kind of *polymorphism*

What is subtyping?
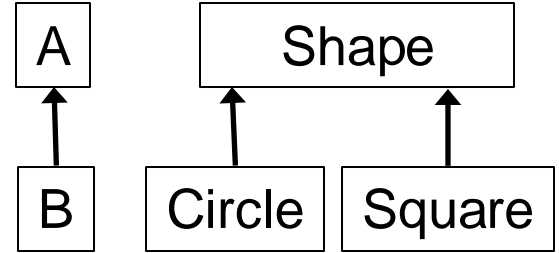
- Sometimes "*every B is an A*"
  - Example:
    - Every Circle or Square is a Shape

A → B

Shape ← Circle, Square

- Subtyping expresses this
  - "*B is a subtype of A*" means: "every object that satisfies the rules for a B also satisfies the rules for an A"

- Goal: code written using A's specification operates correctly even if given a B
  - Plus: clarify design, share tests, (sometimes) share code

Subtyping

- A type S is a **subtype** of T, written $S <: T$, when any term of type S can safely be used in a context where a term of type T is expected.

- $S <: T$ means
  - S is more informative than T.
  - the values of type S are a subset of the values of type T.

The Subsumption Rule

$$\frac{G\vdash\ e:S\quad S <:\ T}{G \vdash e:T}$$

(T-Sub)

- This rule tells us that, if S <: T, then every element t of S is also an element of T.

- For example, if we define the subtype relation so that
$$G\vdash\{x:Int,\ y:Int\} <:\ \{x:Int\}$$
then we can use the subsumption rule to derive
$$G\vdash\ \{x=0,y=1\} <:\ \{x:Int\}$$
which is what we need to make our motivating example typecheck.

Subtyping: A Preorder

- The subtype relation is **formalized as a collection of inference rules** for deriving statements of the form $S <: T$, pronounced "S is a subtype of T" (or "T is a supertype of S").

- The subtype relation should always be a **preorder**, meaning that it is reflexive and transitive.

Reflexivity: $S <: S$ (S-REFL)

Transitivity: $$\frac{S <: U \qquad U <: T}{S <: T}$$ (S-TRANS)

Subtyping — Records: Width Subtyping

- Width Subtyping:

$$\{l_i : T_i^{\ i \in 1..n+k}\} <: \{l_i : T_i^{\ i \in 1..n}\} \qquad \text{S-RCDWIDTH}$$

- A **longer record** constitutes a more demanding—i.e., more informative—specification, and so describes a **smaller set** of values.

- Examples:
  - `{x:Int, y:Int} <: {x:Int}`
  - `{x:Int, y:Int, z:Bool} <: {x:Int}`

Quiz

`{x:Int, y:Int} <: {y:Int}`

A. True
B. False

Quiz

$$\{x{:}Int,\ y{:}Int\} <: \{y{:}Int\}$$

A. True
B. False

Subtyping — Records: Depth Subtyping

▶ Depth Subtyping:

$$\frac{\text{for each } i \quad S_i <: T_i}{\{l_i : S_i{}^{i \in 1..n}\} <: \{l_i : T_i{}^{i \in 1..n}\}}$$

S-RCDDEPTH

▶ It is safe to allow the types of **individual fields** to vary, as long as the types of each corresponding field in the two records are in the subtype relation.

▶ Example:

- `{x:{a:Int, b:Int}, y:{m:Int}} <: {x:{a:Int},y:{}}`

Quiz

Which is the subtype of

$$\{ \ x:\{a:Int, \ b:Bool\} \ \}$$

```
A.  {a:Int,b:Bool}
B.  {x:{a:Int}}
C.  {x:{a:Int}, y:{b:Bool}}
D.  {x:{a:Int, b:Bool,c:Int}, y:{d:Int}}
```

Quiz

Which is the subtype of

$$\{x:\{a:Int,b:Bool\}\}$$

```
A. {a:Int,b:Bool}
B. {x:{a:Int}}
C. {x:{a:Int}, y:{b:Bool}}
D. {x:{a:Int, b:Bool,c:Int}, y:{d:Int}}
```

Subtyping Derivations

$$\cfrac{\cfrac{}{\{a:Nat,b:Nat\} <: \{a:Nat\}} \text{ S-RcdWidth}}{\{x:\{a:Nat,b:Nat\},y:\{m:Nat\}\} <: \{x:\{a:Nat\},y:\{\}\}} \quad \cfrac{}{\{m:Nat\} <: \{\}} \text{ S-RcdWidth} \text{ S-RcdDepth}$$

Subtyping — Records: Permutation Subtyping

► Permutation Subtyping: the order of fields in a record does not make any difference to how we can safely use it

$$\frac{\{k_j : S_j \; ^{j \in 1..n}\} \text{ is a permutation of } \{l_i : T_i \; ^{i \in 1..n}\}}{\{k_j : S_j \; ^{j \in 1..n}\} <: \{l_i : T_i \; ^{i \in 1..n}\}}$$

S-RCDPERM

► Example:
- **{c:Unit,b:Bool,a:Int} <: {a:Int,b:Bool,c:Unit}**

- **{a:Nat,b:Bool,c:Unit} <: {c:Unit,b:Bool,a:Nat}**

Quiz

Which rules will we need to build a derivation of the following?

$$\{x:Int,y:Int,z:Int\} <: \{y:Int\}$$

A. S-RCDDEPTH
B. S-RCDWIDTH
C. S-RCDPERM
D. S-TRANS

Quiz

Which rules will we need to build a derivation of the following?

$$\texttt{\{x:Int, y:Int, z:Int\} <: \{y:Int\}}$$

     A. S-RCDDEPTH
     B. S-RCDWIDTH
     C. S-RCDPERM
     D. S-TRANS

Subtyping — Functions

- Functions can be passed as arguments to other functions, we must also give a subtyping rule for function types

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \rightarrow S_2 <: T_1 \rightarrow T_2} \qquad \text{S-ARROW}$$

- Notice that the sense of the subtype relation is **reversed** (contravariant) for the **argument types** in the left-hand premise, while it runs in the **same direction** (covariant) for the **result types** as for the function types themselves.

Subtyping — Functions

▸ **Intuition**

- Let's say I have a Java function, f, which takes a Cat object and returns an Animal. What are the subtypes of this function? Well, if it takes a Cat then I can certainly replace this function with one that takes an Animal. Likewise, if it returns an Animal then I can certainly replace this function with one that returns a Cat (or Dog). Therefore, I conclude that…

$$(\text{Animal} \rightarrow \text{Cat}) \;<:\; (\text{Cat} \rightarrow \text{Animal})$$

$$(\text{Animal} \rightarrow \text{Dog}) \;<:\; (\text{Cat} \rightarrow \text{Animal})$$