Meeting 29: Subtyping

Today
- Subtyping
- Story: Java's Covariant Arrays
- Object-Oriented Programming and Dynamic Dispatch

Reminders
- Teaching Project, due tomorrow
- Final Exam Review Thu
* driven by you *

Where are we going?

If $T$ is a subtype of $U$, then any expression of type $T$ can be used in a context that expects an $U$. This is called subsumption.

Polymorphism = 1 piece code that/operates over many "forms" = types

parametric polymorphism = "type parameters"

```python
def id[T](x: T): T = x
```

generics (Java, Go)
Widening conversions "upcasts": convert between types respecting subtyping.

Narrowing conversions "downcasts": convert counter to subtyping.

Subclassing in class-based object-oriented languages mix width subtyping with code inheritance and dynamic method dispatch "runtime" "overriding methods".

class D extends C E . . . Z

Is Mystery Broken?

1. Why did type checker complain?
2. Why do we think it should not complain?
VAR \( x : [1 \text{ to } 10] \)

\[ x \gets 1 \]

1. Rule for assignment says
   \[ \text{Type(LHS)} = \text{Type(RHS)} \]
   \[ x : [1 \text{ to } 10] \]
   \[ 1 : \text{INTEGER} \]

2. Too "why" — it doesn't understand
   subtyping relationship between
   \[ [1 \text{ to } 10] \subset \text{INTEGER} \]

Intuitively
   \[ [1 \text{ to } 10] \subseteq [1 \text{ to } 10] \subset \text{INTEGER} \]
Subset Interpretation of Subtyping

\[ T <: U \]

\( T \) is a subtype of \( U \).

For any type \( U \)

\[ U <: U \]

\( U \) subtyping is reflexive.

Subtyping is transitive.

If \( S <: T \) and \( T <: U \)

then \( S <: U \).

What is subtyping useful for?

Subtyping polymorphism

\[ \text{void } f(U a) \{ \]

\( T <: U \)

\[ \} \]

this fun also works for \( T \)'s
\( T \leftarrow t \)
\( U \leftarrow u \)
\( u : = t \)

VAR \( x : [1 \text{ to } 10] \)
VAR \( y : \text{INTEGER} \)

\( x := 2 \) \( \text{ok} \) \([2 \text{ to } 2]\) \( x : [1 \text{ to } 10] \)
\( y := 2 \) \( \text{ok} \) \([2 \text{ to } 2]\) \( y : [1 \text{ to } 10] \)
\( x := 200 \) \( \text{not ok} \) \([200 \text{ to } 200]\) \( x : [1 \text{ to } 10] \)
\( y := 200 \) \( \text{ok} \) \([1 \text{ to } 10]\) \( y : [1 \text{ to } 10] \)
\( x := y \) \( \text{not ok} \)
\( y := x \) \( \text{ok} \)

Q: When is this assignment ok?
\( T = U \) \( \text{ok} \)
\( U < T \) \( \text{unsafe} \)
\( T < U \) \( \text{safe} \)

\( T \leftarrow U \) \( \text{ok} \)
\( U \leftarrow T \) \( \text{narrowing downcast} \)
\( T \leftarrow U \) \( \text{widenable upcast} \)

Q: Allow widening conversions in assignment
\( 2 : [2 \text{ to } 2] \)

Which assignments are allowed?
we may break type safety by "bad" definitions of subtyping

\[
\begin{align*}
\text{INT} \leq \text{INT} & \leq \text{INT} \\
\text{INT} & \leq \text{INT} \Rightarrow \text{INT}
\end{align*}
\]

How do make \( \text{INT} \leq \text{INT} \) type check just by adding a subtyping relationship?

When we define subtyping, we have to be careful.

Plan
- Define rules for base types based on subset interpretation

\[
\begin{align*}
\text{Short} & \leq \text{Int} \\
[a \leq b] & \leq b \text{ Int}
\end{align*}
\]
- Define rules for type constructors

**Subranges**

\[
\frac{a \geq c \quad b \leq d}{[a \to b] \prec [c \to d]}
\]

\[
[1 \to 1] \prec [0 \to 15]
\]

**Pairs**

\[(e_1, e_2) : T_1 \times T_2\]

\[
\frac{T \prec U \quad T' \prec U'}{T \times T' \prec U \times U'}
\]
Records

$$T_i <: T_i' \quad \text{and} \quad n \leq m$$

$$\exists f_1 : T_1, \ldots, f_n : T_n \exists \langle f_1 : T_1', \ldots, f_m : T_m' \rangle$$

$$\{ a : \text{Int} \}$$

$$\Rightarrow$$

$$\{ a : \text{Int}, b : \text{Int} \}$$

width

subtyping

OO