CSCI 742 - Compiler Construction

Lecture 25

More Subtyping Rules

Instructor: Hossein Hojjat

March 28, 2018

Recap: Parametrized Types

- Suppose we know that $S <: T$ (S is a subtype of T)
- Given a parameterized type constructor TYCON$[T]$, there are three possibilities for the relationship between TYCON$[S]$ and TYCON$[T]$

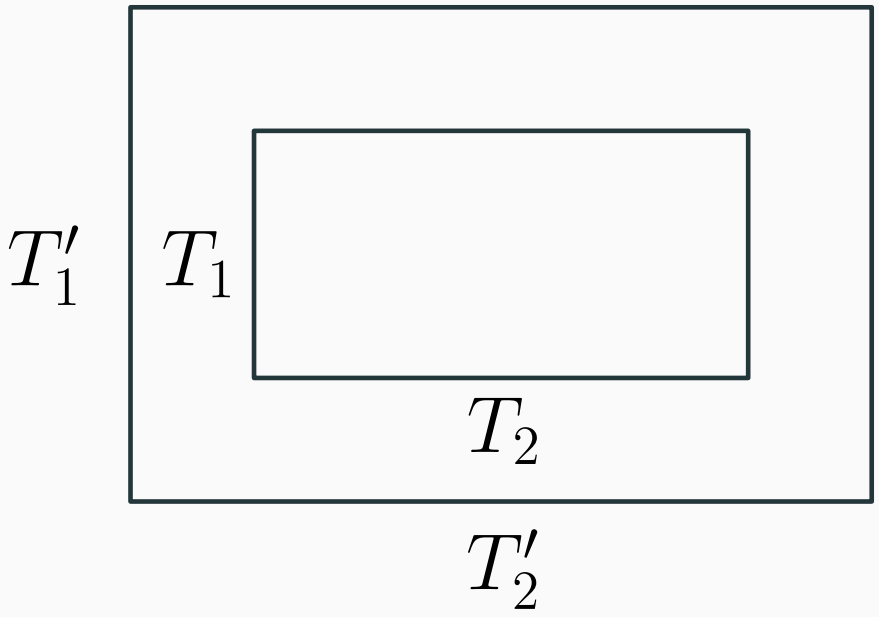
1. **Invariant:** TYCON$[S]$ and TYCON$[T]$ are unrelated
2. **Covariant:** TYCON$[S] <:$ TYCON$[T]$
3. **Contravariant:** TYCON$[S] :>$ TYCON$[T]$

Recap: Analogy with Cartesian Product

- Covariant subtyping for Product of types

$$\frac{T_1 <: T_1' \qquad T_2 <: T_2'}{T_1 \times T_2 <: T_1' \times T_2'}$$

$$\frac{T_1 \subseteq T_1' \qquad T_2 \subseteq T_2'}{T_1 \times T_2 \subseteq T_1' \times T_2'}$$



Subtyping for Function Types

- Is function subtyping a covariant rule?

$$\frac{S_1 <: T_1 \qquad S_2 <: T_2}{S_1 \to S_2 \;<:\; T_1 \to T_2} \quad (???)$$

Subtyping for Function Types

- Recall $\texttt{pos} = \{1, 2, \cdots\}$ (not including zero)

```
pos div (pos x) {
  return 10 / x;
}
```

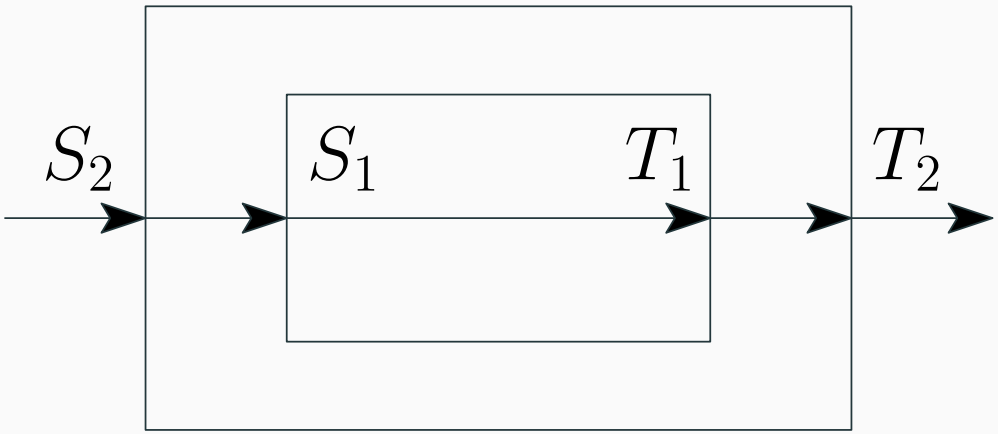
- A covariant rule for subtyping of function types allows the undesired function call `div(0)`

$$\dfrac{\dfrac{\Gamma \vdash \texttt{div} : \texttt{pos} \to \texttt{pos} \qquad \dfrac{\texttt{pos} <: \texttt{int} \qquad \texttt{pos} <: \texttt{pos}}{\texttt{pos} \to \texttt{pos} \;<:\; \texttt{int} \to \texttt{pos}}}{\Gamma \vdash \texttt{div} : \texttt{int} \to \texttt{pos}} \qquad \Gamma \vdash 0 : \texttt{int}}{\Gamma \vdash \texttt{div(0)} : \texttt{pos}}$$

Subtyping for Function Types

- Function type $\rightarrow$ is covariant in the result type and contravariant in the argument type

$$\frac{S_2 <: S_1 \qquad T_1 <: T_2}{S_1 \rightarrow T_1 \;\; <: \;\; S_2 \rightarrow T_2}$$



Exercise

Does the following program type-check correctly? (`int` <: `double`)

```
boolean f(double x) { ... }
int g(int x, double y) { ... }
void h () {
  int x = 10;
  double y = 5.5;
  f(x);
  y = g(x,x);
}
```

Subtype Relation for Arrays

- Java permits covariant subtyping for arrays

$$\frac{S <: T}{S[\,] <: T[\,]}$$

- Bad design, requires run-time checks

```
String[] strings = new String[1];
Object[] objects = strings;
objects[0] = new Integer(0);
```

Exception in thread "main" java.lang.ArrayStoreException: java.lang.Integer

Subtype Relation for Arrays

- What about the contravariant version?

$$\frac{S <: T}{T[\,] <: S[\,]}$$

```
float[] x = new float[1];
int[] y = x; // Use subtyping here.
x[0] = 1.23;
int z = y[0]; // Not an int!
```

Subtype Relation for Arrays

$$\frac{S <: T}{S[\,] \;\; ? \;\; T[\,]}$$

- Correct rule: Array type constructor is invariant
- Only reflexivity applies to array types
- Java and many other "practical" languages use the covariant rule
- Run-time type errors (exceptions) are possible!

Subtyping for Classes

Subtype

- A is a subtype of B ($A <: B$) iff an instance of A can replace an instance of B in any context

Subclass

- Factor out repeated code
- To create a new class, write only the differences

- Java and many OO languages merge these notions for classes
- If `A extends B` (either immediately or transitively) then `A` $<:$ `B`

Subtyping for Classes

- What is the output of the following code?

```
class A {
  int m(long x) { return 0; }
}
class B extends A {
  double m(int x) { return 10; }
}
B b = new B();
System.out.println(b.m(5));
```

Subtyping for Classes

- What is the output of the following code?

```
class A {
  int m(long x) { return 0; }
}
class B extends A {
  double m(int x) { return 10; }
}
B b = new B();
System.out.println(b.m(5));
```

answer: `10.0`

- Function `m` in `B` is an overloading of `m` in `A`: both functions exist in `b`
- Based on type of passed argument, one of the `m` functions is selected at compile time

Subtyping for Classes

- What is the output of the following code?

```
class X {}
class Y extends X {}
class Z {}
class A {
  X m(int x) {
      System.out.println("a"); ...  }}
class B extends A {
  Z m(int x) {
    System.out.println("b"); ...  }}
 new B().m(5);
```

Subtyping for Classes

- What is the output of the following code?

```
class X {}
class Y extends X {}
class Z {}
class A {
  X m(int x) {
      System.out.println("a"); ...  }}
class B extends A {
  Z m(int x) {
    System.out.println("b"); ...  }}
 new B().m(5);
```

answer:

- This program has type error!
- A subclass can only override a method with a covariant return type

Subtyping for Classes

- What is the output of the following code?

```
class X {}
class Y extends X {}
class Z {}
class A {
  X m(int x) {
      System.out.println("a"); ...  }}
class B extends A {
  Y m(int x) {
    System.out.println("b"); ...  }}
 new B().m(5);
```

Subtyping for Classes

- What is the output of the following code?

```
class X {}
class Y extends X {}
class Z {}
class A {
  X m(int x) {
      System.out.println("a"); ...  }}
class B extends A {
  Y m(int x) {
    System.out.println("b"); ...  }}
 new B().m(5);
```

answer: b