Function Subtypes
Overloading, coercion
Top type, Bottom Type

CS 329 Lecture 5

Aug 6, 2007

Rushikesh K. Joshi

Function Types

float f (int x) {..} has type
int-->float

int g (int x) { ...} has type
int --> int

When can we say that a function type is a subtype of another function type?

The Subsumption rule revisited

$$\frac{v:S,\ S<:T}{v:T}$$

By the above rule of subsumption associated with subtype relation <: ,
the values of a subtype can be used safely as values of the (super)type.

Applying the rule to functions,
if (type of g) <: (type of f), we can use g safely wherever type of f is expected. Consider the program given below:

An example

```
int f (int x) {..}
main () {
      int v;
      int x;

                ...
                x = f (v);
                ..
}
```

In the above program when can we use another function g in place of int f(int) in a type-safe manner?

Consider g to be one of the following and find out which of these will be safe replacements for f in the above program?:

- int --> int
- float --> int
- int --> float
- float --> float

An example ..

```
int f (int x) {..}
main () {
      int v;
      int x;

                ...
                x = f (v);
                ..
}
```

We can see that if g defines its input parameter to be float or int, there will be no problem in accepting an input parameter v which is defined as int in the program.

However, if g returns a float type, it will result in loss of information when the return value gets assigned to variable x which has type int.

An example ..

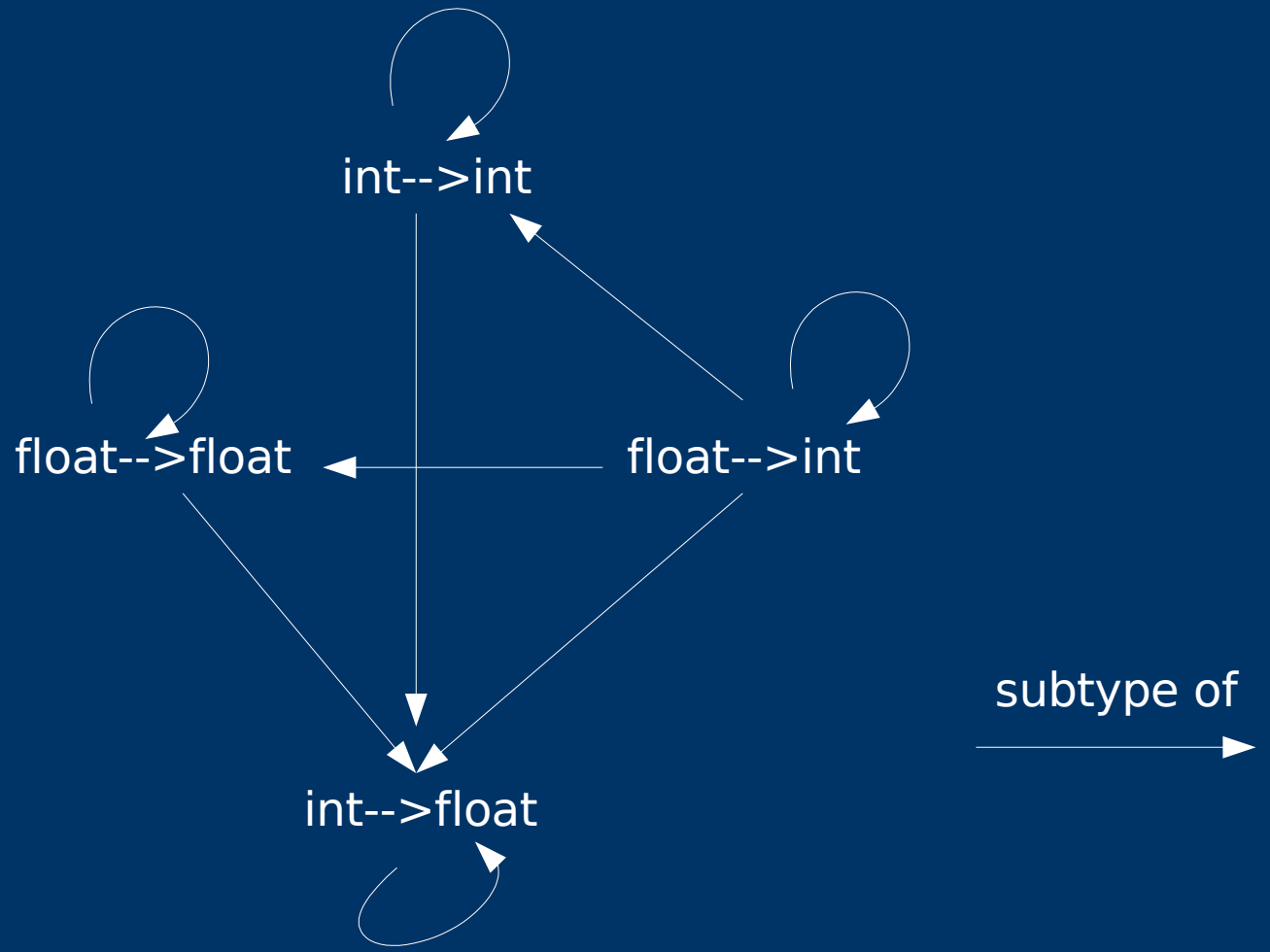
```
int f (int x) {..}
main () {
      int v;
      int x;

                ...
                x = f (v);
                ..
}
```

Both the below functions will be type-safe substitutions for int f(int).

```
int g (int)
int g (float)
```

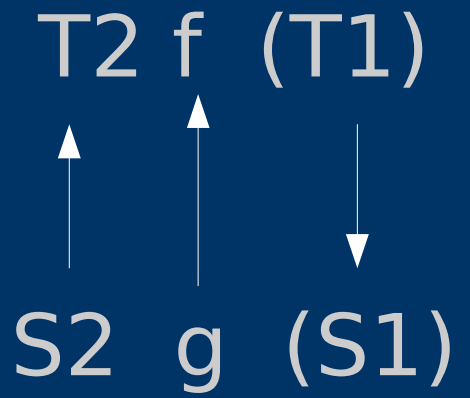
you can see that as long as there is no assignment of a value of a supertype to a variable of subtype, the usage is type safe. This safety condition can be observed in the case of types of input parameters and return results in the above two functions when they are used in place of int f(int).

Type-subtype relations among four functions



The function subtyping rule

input parameters contravariant
output result covariant



Overloading

10 + 2.3
10+ 2
2.3 + 10
2.3 + 2.4

Operator '+' is a function
+: T1 X T2 --> T3

The implementation of operator + is internally provided by the language environment.
Considering the above four possible usages, what can we say about the Type of this internal function '+'. In other words, what should be the signature of this internal function '+'?

What's overloading?

The multiple apparent definitions of a function results in overloading of the function.

The name of the function is the same, but the same name can work with multiple signatures. That is to say that the function name is overloaded.

In the above case, + is overloaded with four possible signatures. However, the language may resolve overloading by using one of the plans discussed below.

Resolving overloading, Plan A

The language may use a single function
float + float --> float
and implicitly type-cast integers to floats and back if needed

The process of implicit type casting is called '**coercion**'

Note that this function is not really a super-type of all other functions. Its
working relies on implicit coercion, and on correct use of types in the program for coercion to work correctly.

int i = 10 + 2 will work correctly as
int i = (int) ((float) 10 + (float) 2) with the typecasts implicitly done.

but int i = 10 + 2.3 will result in loss of accuracy since the lvalue type has been chosen incorrectly as int.

Thus, overloading of 4 signatures can be completely eliminated with the help of just one signature and the use of coercion wherever required.

Resolving overloading, Plan B

use two overloaded functions

float + float --> float
int + int --> int

and implicitly type-cast (coerce) integers to floats and back if needed, if one of the parameters is an int value

Thus, in this case, overloading of 4 signatures is resolved into overloading of 2 signatures with the help of coercion

Resolving overloading, Plan C

use four different functions

float + float --> float
int + int --> int
int + float --> float
float + int --> float

and select the one with exact matching signature.

Thus, in this case, there is no coercion, and full overloading is carried forward into implementation

Top Type

The type of which every other type is a subtype

example: Object type in Java

A value of any type can be used wherever a value of the Top type is expected

Bottom Type

A type of which the values can also be used as values of all other types.

e.g. Type NULLT having a single value NULL.