UNIVERSITY OF Southampton
School of Electronics and Computer Science

COMP2212
PROGRAMMING LANGUAGE CONCEPTS

Julian Rathke and Pawel Sobocinski

SUBTYPING

WHY IS SUBTYPING A GOOD IDEA?

- Suppose that we are in a language of functions and record types.
  - Consider the expression `( fun ( r : { x : int } ) -> r . x ) { x = 2 , y = 3 }`
- The function, call it F, just needs a record with a field labelled x, but the argument also has a field labelled y

only 1 parameter (x) is being passed in
(even though this function can take in x and y)

- Is this a problem?

$$\frac{\vdash F : \{x : \mathsf{Int}\} \rightarrow \mathsf{Int} \qquad \vdash \{x = 2, y = 3\} : \{x : \mathsf{Int}, y : \mathsf{Int}\}}{\Gamma \vdash F\{x = 2, y = 3\} : \mathsf{Int}}$$

- The declared argument type of the function and the actual type of the argument need to match for this rule to be a valid instantiation.
- This expression is **not** well-typed. Even though it seems perfectly sensible.

OVER SPECIFYING TYPE PARAMETERS

generic:
provide a way to create classes, interfaces, and methods
- operate on a parameterised type
- specify the type to be used at compile time
- can create classes and methods that are type safe and reusable

- There are occasions where type systems forces us to be too specific.
  - e.g. example on previous slide
  - function F morally needs an argument of **any** type that has **at least** field `x:Int`
  - another example: a function that returns the length of a array doesn't need to care about what type of elements are in the array.
- What we are looking for in examples such as the above is some sort of generic type, or **polymorphic** type for the type of the parameters to functions.
- Many mainstream programming languages support generics or polymorphism.
- They do so in different ways - in terms of both specification and implementation.

POLYMORPHISM

- 'Polymorphic' literally means "many shaped"
- A polymorphic function may be applied to many different types of data
- There are different varieties of polymorphism:
  - **Parametric polymorphism** (C++ Templates, Java Generics, OCaml)
  - **Subtype polymorphism** (Obj Oriented, C++, Java etc)
  - **Ad hoc polymorphism** (overloading of functions and methods)
- The latter, while useful, is often not so interesting as it simply boils down to clever naming schemes that include types for internal representation of functions and methods.
- The former is a very rich topic that is closely related to type inference and unification a la ML.
- This lecture is about subtype polymorphism

SUBTYPING

- We could think of types as (structured) sets and simply say that a type T is a subtype of U if, in their interpretations as sets [[T]] is a subset of [[U]].
  - This is a nice general definition but it doesn't give us a convenient *syntactic* description of subtyping.
- Two other strong possibilities are evident :

- We could look at the capabilities of the types and say that T is a subtype of U if every operation that can be performed on U can also be performed on T.
  - This definition incorporates lots of structural properties of the types. E.g., pairs must be subtypes of pairs because of the projection operations.
  - This is called **structural subtyping**

- We could explicitly declare what types we want to be subtypes of others and then make sure that any operations valid on a supertype are valid on the subtype.
  - This is the approach taken in object oriented languages, via **inheritance**. It is often called **nominal subtyping**.

SUBSUMPTION AND THE SUBTYPE RELATION

- Either way we look at it, we can extract the following property of subtyping:
  - If T is a subtype of U then every value of type T can also be considered as a value of type U.
  - This property is called **subsumption**.
  - It can be formalised in the following very general type rule

T is a subtype of U

$$\frac{\vdash E : T \qquad T <: U}{\vdash E : U}\ \textsc{TSub}$$

  - this relies on the subtyping relation `T <: U` between types.
  - The same rule is used for both structural and nominal subtyping systems.

- So the obvious next question is how to define the subtyping relation.
- This is where the structural and nominal subtyping systems differ greatly.

NEXT LECTURE: NOMINAL SUBTYPING