# Programming Language Design 2024
# Polymorphism

Hans Hüttel

27 February 2024

1. *(20 minutes)* Give a precise definition of a subtype ordering $\sqsubseteq$ on interval types and prove that it is a partial order.

2. *(25 minutes)* We can define a type constructor `Association` on an interval types and some other type. Values of type `Association (T1,T2)` are lists of pairs whose first elements are values of type `T1` and whose second elements are values of type `T2` and such that all first elements are distinct.

   As an example, the list

   ```
       [ (-3,'f'), (0,'f'),(2,'e'),(4,'d'),(7,'c'),(10,'b'),(12,'a')
         ]
   ```

   is a value whose type is `Association(DanishGrades,Char)`.

   We would now like to extend the subtype ordering to types of the form `Association(T1,T2)` where `T1` and `T2` are interval types, where we assume the subtype ordering $\sqsubseteq$ on interval types that you defined in the solution to the subproblem above.

   The type constructor `Association` takes two types as arguments. Should the constructor be covariant or contravariant wrt. the subtype ordering $\sqsubseteq$ for its first argument? Should it be covariant or contravariant wrt. the subtype ordering $\sqsubseteq$ for its second argument? You must justify your answers.

3. *(30 minutes)* Use the Hindley-Milner algorithm for type inference on the append function as defined in Haskell (in its uncurried version) as

   ```
   append ([],xs) = xs
   append ((x:xs), (y:ys)) = x : (append (xs,ys))
   ```

4. *(25 minutes)* A Haskell programmer is trying to write a function `bingo` that will take any function `f` and any list and produce the value that we get by combining all elements in the list by applying the function `f`.

   The intention is that e.g. if `f` is addition, we should get

   ```
   bingo ((+),[1,2,3,4]) = 10
   ```

   because $1 + 2 + 3 + 4 = 10$.

   The Haskell programmer writes

   ```
   bingo (f, x) = x
   bingo (f, (x:xs)) = f(x, bingo(f,xs))
   ```

   The intention of the first line is that it should capture the case of the empty list – there is no meaningful value to return, so we return the empty list.

   The Haskell programmer gives the definition of `bingo` to the Haskell system and the type inference algorithm does not complain. But when the programmer tries to call the `bingo` function to find `bingo ((+),[1,2,3,4])`, the Haskell interpreter suddenly becomes extremely unhappy and says that there is a type error in the function call.

   Explain, using the Hindley-Milner type inference algorithm, what is wrong. *Hint:* The type of `bingo` is not what the programmer thinks it is. What do you think the programmer would expect it to be?

   How would you fix the definition of `bingo`?