UNIVERSITY OF Southampton
School of Electronics and Computer Science

COMP2212 PROGRAMMING LANGUAGE CONCEPTS

Julian Rathke and Pawel Sobocinski

STRUCTURAL SUBTYPING

STRUCTURAL SUBTYPING

- Relies purely on the structure of the type to define the subtyping relation.
- First, the relationship between base, or primitive types, needs to be specified.
  - For example, one could want **`short <: int`**, **`float <: double`**
- Then structure determines the rest. For example, a subtype of the pair type **`T x U`** is a pair of subtypes of **`T`** and **`U`** separately.

$$\frac{T_1 <: T_2 \qquad U_1 <: U_2}{T_1 \times U_1 <: T_2 \times U_2}\ \textsc{SubPair}$$

- Record types are a generalisation of pair types, and the subtype relation on records generalises in an interesting way too.
- Record types don't rely on syntactic positioning in the values for their indexing. This means we can write a record with some fields missing and have it be perfectly well formed as a record value (of another type). We can't do the same with a tuple.

STRUCTURAL RECORD SUBTYPING

- The rules for subtyping on records can be built up in stages.
- First we have the generalisation of what we saw above for pairs.
- This is called **depth** subtyping for records:

if not talking about parent (supertype) and child (subtype), then it's depth

$$\frac{T_i <: U_i \qquad 1 \leq i \leq n}{\{l_1 : T_1, \dots l_n : T_n\} <: \{l_1 : U_1, \dots, l_n : U_n\}}\textsc{SubRecDepth}$$

- Then we have the notion of **width** subtyping in which there may be extra fields in the subtype:

$$\frac{}{\{l_1 : T_1, \dots, l_n : T_n, l_{n+1} : T_{n+1}\} <: \{l_1 : T_1, \dots l_n : T_n\}}\textsc{SubRecWidth}$$

- And finally, we allow re-ordering of the listed fields:

$$\frac{\sigma \text{ a permutation of } 1 \dots n}{\{l_1 : T_1, \dots l_n : T_n\} <: \{l_{\sigma(1)} : T_{\sigma(1)}, \dots, l_{\sigma(n)} : T_{\sigma(n)}\}}\textsc{SubRecPerm}$$

- Not all languages adopt all of these principles. Indeed, Java does not allow depth subtyping. Methods or fields in a subclass cannot have subtypes of that with which they are declared in the supertype.

EXAMPLE OF RECORD SUBTYPING

$$
\dfrac{\dfrac{\dfrac{}{\{\,x : \mathrm{int}\,,\, y : \mathrm{int}\,,\, z : \mathrm{int}\,\} \;<:\; \{\,x : \mathrm{int}\,,\, y : \mathrm{int}\,\}}\,\text{SubRecWidth} \qquad \dfrac{}{\{\,x : \mathrm{int}\,,\, y : \mathrm{int}\,\} \;<:\; \{\,x : \mathrm{int}\,\}}\,\text{SubRecWidth}}{\{\,x : \mathrm{int}\,,\, y : \mathrm{int}\,,\, z : \mathrm{int}\,\} \;<:\; \{\,x : \mathrm{int}\,\}}\,\text{SubTrans} \qquad \dfrac{}{\{\,w : \mathrm{int}\,\} \;<:\; \{\,\}}\,\text{SubRecWidth}}{\{\,S : \{\,x : \mathrm{int}\,,\, y : \mathrm{int}\,,\, z : \mathrm{int}\,\}\,,\, R : \{\,w : \mathrm{int}\,\} \;<:\; \{\,S : \{\,x : \mathrm{int}\,\}\,,\; R : \{\,\}\,\}}\,\text{SubRecDepth}
$$

STRUCTURAL SUBTYPING FOR VARIANTS

- For sum types `T + U`, it is intuitive that any value of type `T` can be considered as also being of type `T + U + V` as the value of type `T` is still just injected in to the sum, albeit in to a larger sum.
- For generalised, variant, types. We have the same notions of width, depth and permutation subtyping as we do for records.
- There is however, a subtle inversion in the rule for width subtyping:

$$\frac{}{\langle l_1 : T_1, \ldots l_n : T_n \rangle <: \langle l_1 : T_1, \ldots, l_n : T_n, l_{n+1} : T_{n+1} \rangle}\textsc{SubVarWidth}$$

- We see that we can inject in to a larger variant type
- The notions of depth and permutations are identical though.

$$\frac{T_i <: U_i \qquad 1 \leq i \leq n}{\langle l_1 : T_1, \ldots l_n : T_n \rangle <: \langle l_1 : U_1, \ldots, l_n : U_n \rangle}\textsc{SubVarDepth}$$

$$\frac{\sigma \text{ a permutation of } 1 \ldots n}{\langle l_1 : T_1, \ldots l_n : T_n \rangle <: \langle l_{\sigma(1)} : T_{\sigma(1)}, \ldots, l_{\sigma(n)} : T_{\sigma(n)} \rangle}\textsc{SubVarPerm}$$

NEXT LECTURE: VARIANCE