Exercise 5: Type Checking

Explain in your own words the difference between name equivalence and structural equivalence in type checks. Why does Java (and MicroJava) use name equivalence in most cases, but structural equivalence for arrays?

Solution

With *name equivalence*, two types are considered to be the same if they are described by the same type name. In the compiler, this only requires a check whether they are represented by the same structure node. The type check therefore boils down to a comparison of whether two type pointers point to the same node (a.type == b.type), which is simple and efficient.

With *structural equivalence*, two types are considered to be the same if they have the same structure. For arrays, this requires a check whether they have the same element type. For classes, it is necessary to check whether they have the same number of fields and whether the types of the corresponding fields are structurally equivalent, which amounts to a recursive comparison. Although structural equivalence is more powerful than name equivalence, it is more complex and requires a more complicated and also more inefficient comparison of structured types.

For reasons of simplicity (and also better comprehensibility), most modern programming languages therefore use name equivalence. In Java (and also in MicroJava), however, structural equivalence is used for arrays, because array types in Java cannot be described by a type name (e.g. `PersonArray`), but only by a type structure (e.g. `Person[]`).