# Julia Subtyping: A Rational Reconstruction

FRANCESCO ZAPPA NARDELLI, Inria and Northeastern U.
JULIA BELYAKOVA, Czech Technical U. in Prague
ARTEM PELENITSYN, Czech Technical U. in Prague
BENJAMIN CHUNG, Northeastern U.
JEFF BEZANSON, Julia Computing
JAN VITEK, Northeastern U. and Czech Technical U. in Prague

## THE DIAGONAL RULE, AS SUPPORTED BY C#

Although the design of the diagonal rule employed in Julia may look surprising at first, it turns out that similar rules apply in C# for static method resolution.

```
abstract class B {}
class D1 : B {}
class D2 : B {}

class Program {
    static void diag<T>(T x, T y) {}

    static void nondiag<T>(T x, T y, T[] zs) {}

    static void Main() {
        //diag(new D1(), new D2());             // (1) error
        diag(new D1(), new D1());               // (2)
        nondiag(new D1(), new D2(), new B[5]);  // (3)
    }
}
```

The method `diag` takes two arguments of type `T`: C# static method resolution enforces that the arguments, when `T` is to be inferred, must have the same type. So the method invocation in (1) is incorrect, and raises a type checker error because types `D1` and `D2` are distinct. The method invocation in (2) is instead correct because the arguments are instances of the same type `D1`. As in Julia, variables with invariant occurrences cannot be diagonal, as the variable `T` in the method `nondiag`: the method invocation in (3) is correct.

We remark that Java Generics instead do not support the diagonal rule because of erasure: generic methods are compiled down to regular ones and all occurrences of `T` turn into `Object`. A call to an erased version of a method will thus accept any combination of parameters.