Introducing Scala-like functional interfaces into Java
– Abstract –

Martin Plümicke

Baden-Wuerttemberg Cooperative State University Stuttgart
Department of Computer Science
Florianstraße 15, D–72160 Horb
pl@dhbw.de

April 8, 2015

A lambda expression in Java 8 has no explicit type. The type is determined by the compiler from the context in which the expression appears. This means that one lambda expression can have different types in different contexts.

```
Callable<String> c = () -> "done";
PrivilegedAction<String> a = () -> "done";
```

In the first context for the lambda expression the type `Callable<String>` is determined, while in the second context `PrivilegedAction<String>` is determined.

The determined types are called *target types*. Not all determined target types are correct. The determined target type is correct if the lambda expression is compatible with it [Goe13].
As there are many callback interfaces in the existing Java libraries this approach is very convienient as it avoids writing uncomfortable anonymous inner classes.
But in this approach it is very difficult to use subtypes of functional interfaces. Furthermore the direct evaluation of lambda expressions is very unconvenient. We considered both in [Plü14].
Therefore we introduce a set of special interfaces `Fun*`N, where the subtyping property is changed in comparison to Java. The special interfaces `Fun*`N correspond to functions types in Scala [Ode14].

References

[Goe13] Brian Goetz. State of the lambda, September 2013.

[Ode14] Martin Odersky. *The Scala Language Specification Version 2.9*, May 2014.

[Plü14] Martin Plümicke. Functional Interfaces vs. Function Types in Java with Lambdas – Extended Abstract. In *Tagungsband der Arbeitstagung Programmiersprachen (ATPS 2014)*, volume Vol-1129, pages 146–147. CEUR Workshop Proceedings (CEUR-WS.org), 2014.