Controlling effects

Alexander Kuklev[1,2] ⟨a@kuklev.com⟩

[1]Radboud University Nijmegen, Software Science
[2]JetBrains Research

In many cases, high-order functions such as `sortWith(comparator)` only have meaningful behaviour if their arguments are pure functions. Type-level control over the purity of functions and data is essential to statically prevent nonsensical behaviour and dangerous vulnerabilities.

TODO: Say a couple of words on pure data and constants.

1 Pure data

Pure data types are primitive data types (`Boolean`, `Int`, `Float`, etc.), enums, strings, pure functions, immutable arrays of pure data, as well as (final or sealed) classes and objects where all fields are immutable and pure, and all methods are pure functions. We propose introducing a modifier keyword `pure` for classes and interfaces to require them to be pure and only have pure descendants, as well as for type parameters. Pure data types include algebraic data types:

```
sealed value class List<pure T> {

  object EmptyList : List<Any?>()

  data class NonEmptyList<pure T>(val head : T,
                                  val tail : List<T>) : List<T>
}
```

2 Pure functions

We propose introducing `pure` modifier for functions and function types to forbid them referring to any external objects of not-pure datatype, including global non-pure objects such as `System`, `Runtime`, etc. Now the sort function can require the comparator to be pure:

```
fun <T> Array<out T>.sortWith(comparator : pure Comparator<in T>)
```

It is often desirable to allow semi-purity, e.g. we can allow the `comparator` to use the `Logger`:

```
fun <T> Array<out T>.sortWith(comparator : pure(Logger) Comparator<in T>)
```

It also works beyond function types: `pure(capabilities) T` are precisely the capturing types `T^{capabilities}` of the Scala3 experimental capture checking layer[1] which allows fine-grained control of effects and captured references, see Scoped Capabilities for Polymorphic Effects.[2]

3 Constants

Constant properties must be allowed to have pure types, not only strings and values of primitive data types, as it is currently mandated in Kotlin. Their values should be allowed to contain not only literals and other constants, but also `pure(CompileTime)`-functions applied to literals and constants, where `CompileTime` provides access to the build environment and resource files:

```
const val APP_ENV = CompileTime.getenv("APP_ENV") ?: "DEV"
const val DB_SCHEMA = DbSchema("jdbc:sqlite:./resources/prototypeDb")
```

[1]https://docs.scala-lang.org/scala3/reference/experimental/cc.html
[2]https://arxiv.org/abs/2207.03402