SCALA TYPE-LEVEL OPERATIONS

Matt Bovel @LAMP/LARA, EPFL

October 5, 2022

OUTLINE

INTRODUCTION

ROCKETS EXPLODE

x

TYPES CAN HELP

`x : Long`

LITERAL TYPES

Type inhabited by a single constant value known at compile-time:

```scala
val x: 3 = 3
val y: false = false
val z: "monday" = "monday"
```

See SIP-23 - literal-based singleton types.

PATH-DEPENDENT TYPES

Type inhabited by a single non-necessary-constant term:

```scala
val a: Int = ???
val b: Int = ???

val c: a.type = a
val d: Int = a     // Ok because (a: Int) <: Int
```

```scala
val e: a.type = b // Error: found (b: Int)
                  // but required (a: Int)
```

It is called *path*-dependent because it can refer to nested members as well:

```
object Foo:
    val x: 3 = 3
summon[Foo.x.type =:= 3]
```

Note: instances of the `=:=` type are generated automatically by the compiler when the left hand-side and the right hand-side are both subtypes of each other. Therefore, `summon[X =:= Y]` compiles only if `X` are equivalent `Y`.

DEPENDENT PARAMETERS

Singletons are used to model *equality* between terms.

```
def same(a: Any, b: a.type) = ???
same(3, 3) // Ok
same(3, 4) // Error
```

Writing the same function with a type parameter instead has a different meaning. It asks the compiler to find a `T` such that `3 <: T` and `4 <: T` which is satisfiable using `T = Int`:

```
def same2[T](a: T, b: T) = ???
same2(3, 4) // Ok; T is inferred to be Int
```

DEPENDENT RETURN TYPES

```
def id(x: Any): x.type = x
```

REFINEMENT TYPES

```
class Vec:
  val size: Int

val v: Vec {val size: 2} = new Vec:
  val size: 2 = 2

val vSize: 2 = v.size
```

COMPILE-TIME OPERATIONS

Simple bounded type aliases:

```scala
infix type +[X <: Int, Y <: Int] <: Int
```

With special compiler support for constant-folding:

```scala
import scala.compiletime.ops.int.+

val a: 2 + 2 = 4
```

See Add primitive compiletime operations on singleton types #7628.