SCALA TYPE-LEVEL OPERATIONS

Matt Bovel @LAMP/LARA, EPFL

June 28, 2022

OUTLINE

- Literal types
- Term-reference types
- Dependent parameter types
- Compile-time operations
- Refinement types
- Example: Check `Vec.map` (type params)
- Example: tf-dotty (with abstract dimensions)
- Open problems with normalization
- Match types
- Example: Sized vectors
- Algebraic reasoning
- Example: Check `Vec.map` (refinements)
- Example: strongly-typed `printf`
- Example: regsafe
- Selectable
- Hands-on: strongly-typed CSV (macros)
- Hands-on: implement `Read`
- Thanks!

LITERAL TYPES

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

```
val x: 3 = 3
val y: false = false
val z: "monday" | "tuesday" = "monday"
```

See SIP-23 - literal-based singleton types.

TERM-REFERENCE TYPES

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

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

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

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

DEPENDENT PARAMETER TYPES

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

same(3, 4) // Error
```

```
def same2[T](a: T, b: T) = true
same2(3, 4) // Ok
```

COMPILE-TIME OPERATIONS

Simple bounded type aliases:

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

With special compiler support for constant-folding:

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

val a: 2 + 2 = 4
```

See Add primitive compiletime operations on singleton types #7628.

REFINEMENT TYPES

```
case class Vec(size: Int)

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

val size: 2 = v.size
```

EXAMPLE: SIZED VECTORS

```scala
//> using options "-Xprint:typer"
import scala.compiletime.ops.int.+
def vec(s: Int) = Vec(s).asInstanceOf[Vec {val size: s.type }]
def add(a: Int, b: Int) = (a + b).asInstanceOf[a.type + b.type]


case class Vec(size: Int):
  def sum(that: Vec {val size: Vec.this.size.type}) = vec(size)
  def concat(that: Vec) = vec(add(size, that.size))

val v: Vec {val size: 13} = vec(6).concat(vec(7)).sum(vec(13))
```

Source: `examples/1_vec.scala`

ALGEBRAIC REASONING

```
// Summing x n times is normalized to x * n.
summon[2L * m.type =:= m.type + m.type]
summon[2L * m.type + 2L * m.type =:= m.type + 3L * m.type]
summon[2L * m.type * m.type =:= m.type * 2L * m.type]
```

EXAMPLE: CHECK `Vec.map` (REFINEMENTS)

Live.

EXAMPLE: CHECK `Vec.map` (TYPE PARAMS)

```
import compiletime.ops.int.{+,-}

enum Vec[Len <: Int, +T]:
  case Nil extends Vec[0, Nothing]

  case NotNil[T]() extends Vec[Int, T]

  def ::[S >: T](x: S): Vec[Len + 1, T] = ???
  def tail: Vec[Len - 1, T] = ???
  def head: T = ???
```

See Boruch-Gruszecki, A. (2019). GADTs in Dotty. Slide 16.

EXAMPLE: TF-DOTTY (WITH ABSTRACT DIMENSIONS)

```
val x: Int = 2
val y: Int = 2
val tensor = tf.zeros(x #: y #: SNil)
val res = tf.reshape(tensor, y #: x #: SNil)
```

See github.com/MaximeKjaer/tf-dotty, in particular the implementation of `reshape`.

OPEN PROBLEMS WITH NORMALIZATION

- Exponential explosion and compilation time
- Sub-typing

MATCH TYPES

```
type IsEmpty[S <: String] <: Boolean = S match {
  case "" => true
  case _ => false

}

summon[IsEmpty[""] =:= true]
summon[IsEmpty["hello"] =:= false]
```

See Blanvillain, O., Brachthäuser, J., Kjaer, M., & Odersky, M. (2021). Type-Level Programming with Match Types. 70.

EXAMPLE: STRONGLY-TYPED `printf`

```
//> using scala "3.nightly"
//> using options "-Xprint:typer"
import scala.compiletime.ops.int.{+}
import scala.compiletime.ops.string.{CharAt, Length, Substring}

import scala.Tuple._

type ArgTypes[S <: String] <: Tuple =
  S match {
    case "" => EmptyTuple
    case _ => CharAt[S, 0] match {
```

Source: `examples/2_printf.scala`

See also `printf` in Zig

EXAMPLE: REGSAFE

```
import regsafe.Regex

val date = Regex("""(\d{4})-(\d{2})-(\d{2})""")
"2004-01-20" match
  case date(y, m, d, a) =>
    s"$y was a good year for PLs."
```

See github.com/OlivierBlanvillain/regsafe and Blanvillain, O. (2022). Type-Safe Regular Expressions.

SELECTABLE

Not presented.

```
class MySelectable extends Selectable:
    def selectDynamic(name: String): Any =
        name match

            case "foo" => 1
            case "bar" => "hey"

val s = MySelectable().asInstanceOf[MySelectable {val foo: Int; val bar:
String}]

@main def test() =
```

Source: `examples/3_selectable.scala`

HANDS-ON: STRONGLY-TYPED CSV (MACROS)

Not presented.

```
import scala.quoted.*

object Macro {

  transparent inline def read(file: String): Any =
    ${ impl('file) }

  private def impl(file: Expr[String])(using Quotes): Expr[Any] = {
    import quotes.reflect.*
    val refinementType = Refinement(TypeRepr.of[Object], "hello",
TypeRepr.of[String]).asType
```

Source: `examples/4_macro_refinement/1_macro.scala`

HANDS-ON: IMPLEMENT `Read`

Live.

Code written during the live session: https://github.com/dotty-staging/dotty/commit/7dfde36139bc380ac9c6e0c4f5ee80647006c29e.

For a real-world example of type-generation at compile-time, see FSharp.Data and F# Type Providers.

THANKS!