More ergonomic union types

16 december 2022

Outline

- Brief summary of this long thread
  - `https://contributors.scala-lang.org/t/making-union-types-even-more-useful/4927`
- Next steps?

These slides are here:
`https://github.com/bjornregnell/scala-sip-notes`

Better type inference for union types

Improvements since 3.1.x

- More precise types in 3.2.1 for some cases:
  - `https://github.com/lampepfl/dotty/issues/11449`
- But some problems remains:
  - `https://github.com/lampepfl/dotty/issues/14642`
- ”De-duplication” of union types:
  - `https://github.com/lampepfl/dotty/issues/10693`

Lagom widening

```
scala> var p = true
var p: Boolean = true

scala> val x = if p then 42 else "hello"
val x: Matchable = 42

// why not infer the more precise Int | String  ?
```

Scrap boilerplate by generating match on unions

```
scala> class A(val x: Int)

scala> class B(val x: Int)

scala> val ab: A | B = A(42)

scala> ab.x
-- [E008] Not Found Error: ----------------------------------------
1 |ab.x
^^^^
value x is not a member of A | B
```

- You can fix this with match boilerplate:

```
scala> ab match
   case a: A => a.x
   case b: B => b.x
```

- But the compiler knows statically that both A and B has x
- Proposal: make the compiler synthesize the match

Discussions on member selection

- *Martin Odersky:* One problem here is that union types are supposed to be commutative, but the match is order-dependent, unless we can prove somehow that all alternatives are disjoint.
- *...after some discussions...*
- *Martin Odersky:* Yes, I think you are right, as long as we restrict our focus to member selection. If there is an overlap then dynamic dispatch will provide the right method.

```
scala> extension (x: Int) def size = x
def size(x: Int): Int

scala> List[Int|String](42,"fortytwo").map(_.size)
// value size is not a member of Int | String
```

- *Sébastien Doeraene*: Including extension methods of individual components of the unions is an even bigger can of worms...