By-Name Context Parameters

Context parameters can be declared by-name to avoid a divergent inferred expansion. Example:

```
trait Codec[T]:
  def write(x: T): Unit

given intCodec: Codec[Int] = ???

given optionCodec[T](using ev: => Codec[T]): Codec[Option[T]] with
  def write(xo: Option[T]) = xo match
    case Some(x) => ev.write(x)
    case None =>

val s = summon[Codec[Option[Int]]]

s.write(Some(33))
s.write(None)
```

As is the case for a normal by-name parameter, the argument for the context parameter `ev` is evaluated on demand. In the example above, if the option value `x` is `None`, it is not evaluated at all.

The synthesized argument for a context parameter is backed by a local val if this is necessary to prevent an otherwise diverging expansion.

The precise steps for synthesizing an argument for a by-name context parameter of type `⇒ T` are as follows.

1. Create a new given of type `T`:

   ```
   given lv: T = ???
   ```

   where `lv` is an arbitrary fresh name.

where `lv` is an arbitrary fresh name.

2. This given is not immediately available as candidate for argument inference (making it immediately available could result in a loop in the synthesized computation). But it becomes available in all nested contexts that look again for an argument to a by-name context parameter.

3. If this search succeeds with expression `E`, and `E` contains references to `lv`, replace `E` by

   ```
   { given lv: T = E; lv }
   ```

   Otherwise, return `E` unchanged.

In the example above, the definition of `s` would be expanded as follows.

```
val s = summon[Test.Codec[Option[Int]]](
  optionCodec[Int](using intCodec)
)
```

No local given instance was generated because the synthesized argument is not recursive.

Reference

For more information, see Issue #1998 and the associated Scala SIP.

‹ Implici... Relatio... ›