Unleashing Scalafix potential with custom rules

Scalameta
for everyday tasks

Brice Jaglin
ScalaIO Nantes | February 2024

Currently Staff Engineer
formerly Engineering Director

Scala professional
from 2013 to 2022

Non-sponsored
Scalafix maintainer since 2020

swile

Employee benefits
(meal vouchers and more)

Operations in
France & Brazil

High growth & profitable

Ruby shop on the backend

Hiring senior and staff engineers

Features

Lints & rewrites

```
Ops.scala:7:3: error: [DisableSyntax.var] mutable state should be avoided
var pending = 0
^^^
```

```
 import scala.util.control.NonFatal
- import scala.concurrent.Future
 import scala.util.{Properties, Try}
```

```
complete(isSuccess = true)
```

For users

- A set of battle-tested rules built-in
- Drop-in support for custom rules
- Rule suppression via annotations
- CI-friendly via `--check`
- Dev-friendly via `scalafixOnCompile` & incremental runs (sbt-scalafix)

For rule authors

- Portable thanks to Scalameta
  - Abstract Syntax Trees (like scalafmt)
  - SemanticDB information (like metals)
- Extensible
  - Testkit for “expects tests”
  - On-demand or ahead of time compilation for custom rules

Build tools

- sbt-scalafix
  - Official support, biggest feature set
  - Release cycle in sync with Scalafix
- joan38/mill-scalafix
  - by Joan Goyeau
- cosmicsilence/gradle-scalafix
  - by Marcelo Cenerino
- evis/scalafix-maven-plugin
  - by Evgeny Veretennikov
- Bazel
  - POC by Eugene Yokota
  - semanticdb in rules_scala#1508
- scala-cli
  - No ongoing effort at the moment
  - See VirtusLab/scala-cli#647

Integrations

- Scala Steward
  - Declare rewrite rules to run on bumps
  - Run on the open source instance or on a corporate one
- Metals
  - “Organize imports” driven by Scalafix
  - Command metals.scalafix-run
  - Has its own ExplicitResultTypes
  - No way to preview/select patches nor to see diagnostics in line

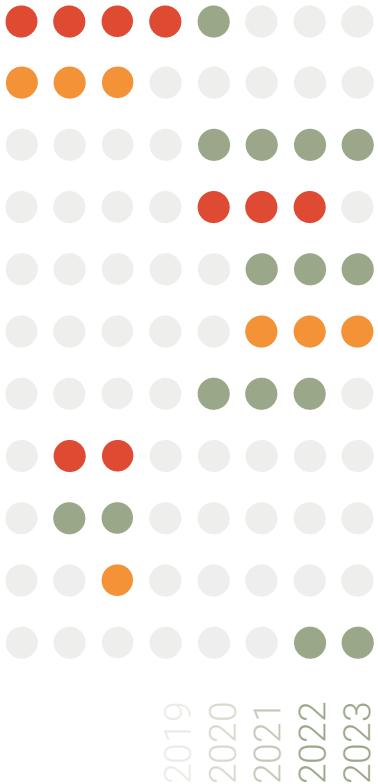
Other active linters & code transformers

- WartRemover
  - Runs as compiler plugin on Scala 2.x & against TASTy on Scala 3
  - Possible to write custom warts
  - Linter only
- scalac Quickfixes / actionable diagnostics
  - Exposed via BSP/LSP
  - Quickfixes must be defined in the compiler (Scala 2.13 and/or Scala3)
- IntelliJ
  - Powerful refactoring actions
  - Custom toolchain, not enforceable on CI

Project activity

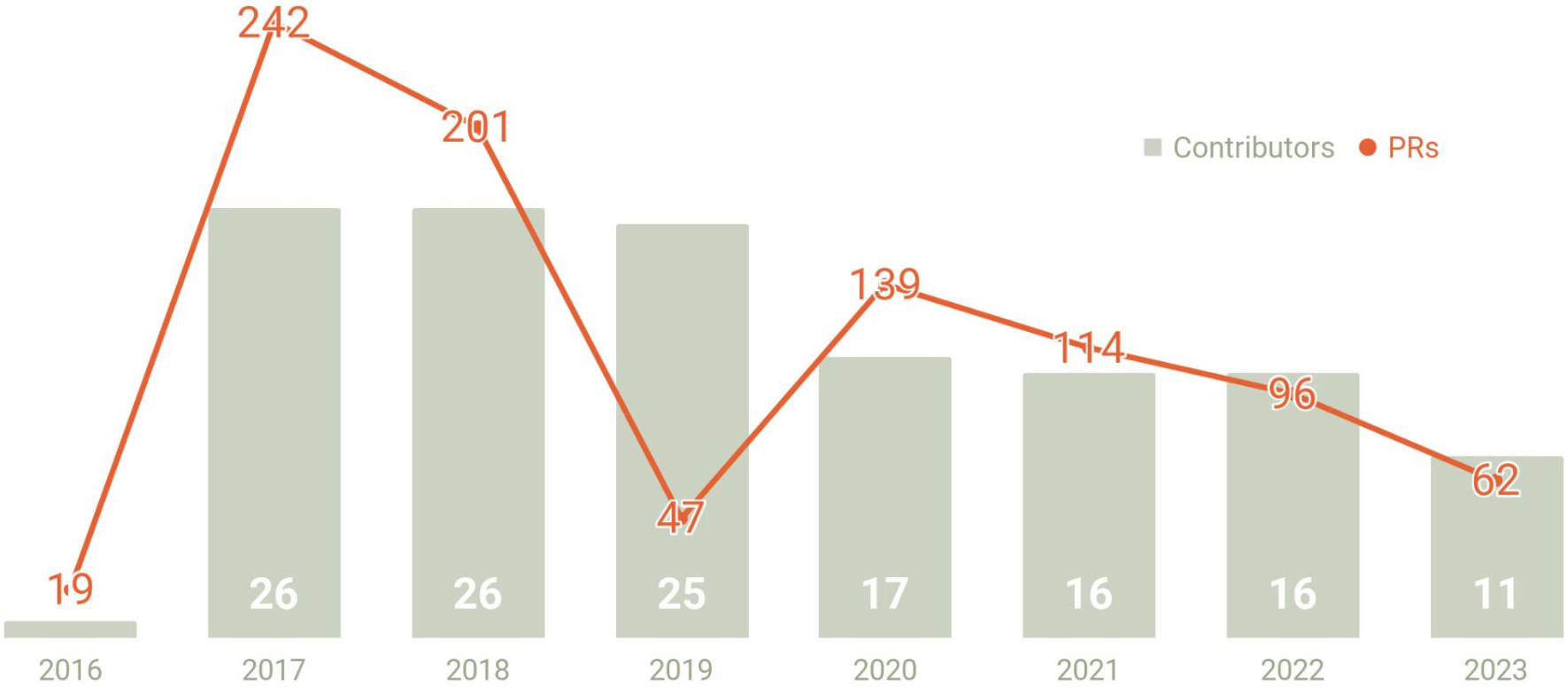
Major direct & indirect contributors & sponsors

Ólafur Páll Geirsson	Scalafix, sbt-scalafix & Scalameta		Scala Center
Eugene Burmako	Scalameta		Twitter
Brice Jaglin	Scalafix & sbt-scalafix		
Meriam Lachkar	Scalafix & sbt-scalafix		Scala Center
Albert Meltzer	Scalameta (for scalafmt)		
Tomasz Godzik	Scalameta (for metals)		VirtusLab
Cheng Lian	`OrganizeImports`		
Guillaume Massé	Scalafix, Scalameta		Scala Center
Gabriele Petronella	Scalafix, Scalameta		
Vladimir Polushin	Scaluzzi		@fommil
Kenji Yoshida	scalafix-rules & scalameta-ast		

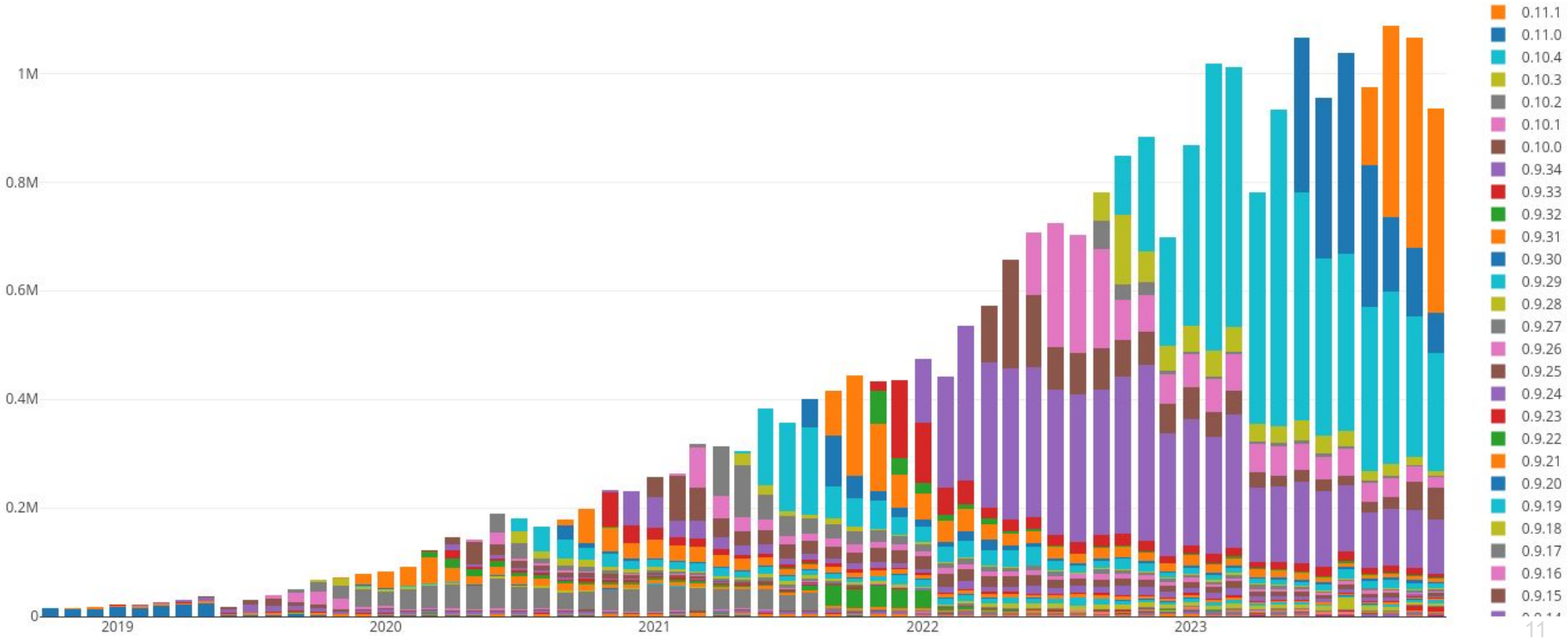


Yearly contributions to scalafix & sbt-scalafix

excluding bots



Monthly Sonatype downloads (scalafix-interfaces)



Under the hood

Topology of a few built-in rules

	Syntactic	Semantic	Semantic with presentation compiler
⚠️ Diagnostics / lints	`DisableSyntax`		
✏️ Rewrites	`RedundantSyntax`	`RemoveUnused`	`Explicit Result Types`

HelloWorld.scala

```scala
package test
import scala.util.Try
object HelloWorld:
  @main def hello =
    println("Hello, world!")
```

parser

```
...
 Importer(
   Term.Select(
     Term.Name("scala"),
     Term.Name("util")
   ),

List(Importee.Name(Name("Try")))
 )
...
```

HelloWorld.scala.semanticdb

```
Occurrences:
[0:8..0:12) <= test/
[1:7..2:12) => scala/
[1:13..2:17) => scala/util/
[1:18..2:21) => scala/util/Try#
...

Diagnostics:
[1:18..1:21) [warning] unused import
```

.scalafix.conf

```
rules = [
 RemoveUnused
 DisableSyntax
]
DisableSyntax.regex = [
 "println"
]
```

Inspecting source files

https://astexplorer.net

AST Explorer | Snippet | Scala | scalameta | Transform | default | Parser: scalameta-4.8.7

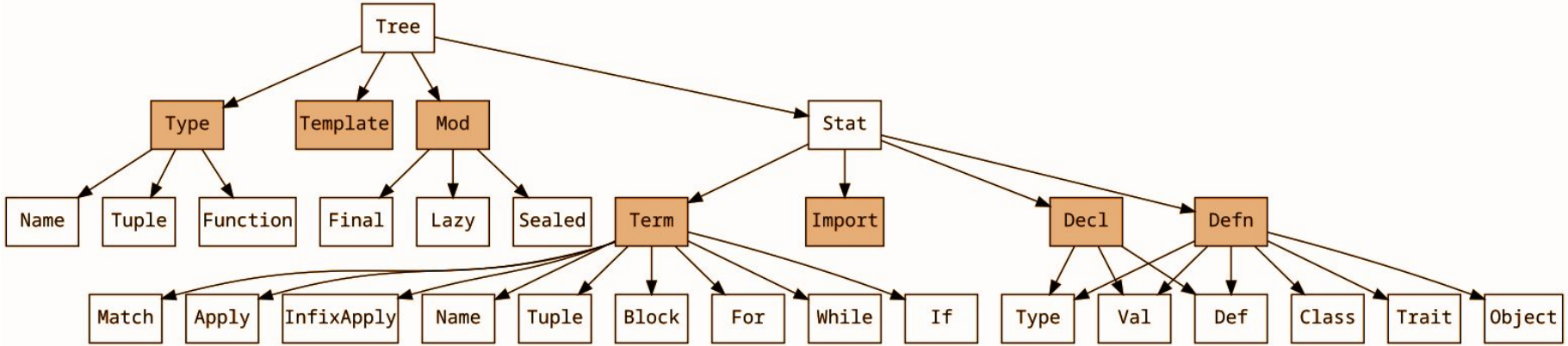
```scala
package test

import scala.util.Try

object HelloWorld:
 @main def hello =
   println("Hello, world!")
```

Tree | JSON | 18ms

☐ Autofocus ☐ Hide methods ☑ Hide empty keys ☐ Hide location data

```
- Import {
  + pos: {start, end}
  - importers: [
    - Importer {
      - pos: {
          start: 21
          end: 35
        }
      + ref: Term.Select {pos, qual, name}
      - importees: [
        - Importee.Name {
          + pos: {start, end}
          - name: Name.Indeterminate {
            + pos: {start, end}
              value: "Try"
            }
        }
      ]
```

Scalameta Tree type hierarchy



```scala
package test
import scala.util.Try
object HelloWorld:
  @main def hello =
    println("Hello, world!")
```

```scala
Seq(
 Token.BOF,
 Token.KwPackage,
 Token.Space,
 Token.Ident("test"),
 Token.LF,
 Token.KwImport,
 Token.Space,
 Token.Ident("scala"),
 Token.Dot,
 Token.Ident("util"),
 Token.Dot,
 Token.Ident("Try"),
 Token.LF,
 Token.KwObject,
 Token.Space,
 Token.Ident("HelloWorld"),
 Token.Colon,
 Token.LF,
 Token.Space,
```

```scala
Source(
 List(
   Pkg(
     Term.Name("test"),
     List(
       Import(
         List(
           Importer(
             Term.Select(
               Term.Name("scala"),
               Term.Name("util")
             ),
             List(
               Importee.Name(
                 Name("Try")
               )
             )
           )
         )
       ),
       Defn.Object(
         Nil,
         Term.Name("HelloWorld"),
         Template(
           Nil,
           Nil,
           Self(Name.Anonymous(), None),
           List(
             Defn.Def(
```

```scala
package test
import scala.util.Try
object HelloWorld:
  @main def hello =
    println("Hello, world!")
```

```scala
Seq(
 Token.BOF,
 Token.KwPackage,
 Token.Space,
 Token.Ident("test"),
 Token.LF,
 Token.KwImport,
 Token.Space,
 Token.Ident("scala"),
 Token.Dot,
 Token.Ident("util"),
 Token.Dot,
 Token.Ident("Try"),
 Token.LF,
 Token.KwObject,
 Token.Space,
 Token.Ident("HelloWorld"),
 Token.Colon,
 Token.LF,
 Token.Space,
```

```scala
Source(
 List(
   Pkg(
     Term.Name("test"),
     List(
       Import(
         List(
           Importer(
             Term.Select(
               Term.Name("scala"),
               Term.Name("util")
             ),
             List(
               Importee.Name(
                 Name("Try")
               )
             )
           )
         )
       ),
       Defn.Object(
         Nil,
         Term.Name("HelloWorld"),
         Template(
           Nil,
           Nil,
           Self(Name.Anonymous(), None),
           List(
             Defn.Def(
```

Traversing a Scalameta tree

```scala
def collect[T](
  fn: PartialFunction[Tree, T]
): List[T] = {

 val liftedFn = fn.lift
 val buf = ListBuffer[T]()

 def traverser(tree: Tree): Unit = {
   liftedFn(tree).foreach(buf += _)
   tree.children.foreach(traverser)
 }

 traverser(tree)
 buf.toList
}
```

```scala
Importer(
  Term.Select(
    Term.Name("scala"),
    Term.Name("util")
  ),
  List(
    Importee.Name(
      Name("Try")
    )
  )
)
```

Scalameta quasiquotes

```scala
import scala.meta._

object Quasiquotes {
 importer"scala.util.Try"
}
```

```scala
[[syntax trees at end of typer]] // Quasiquotes.scala
package <empty> {
 import scala.meta._;
 object Quasiquotes extends scala.AnyRef {
   def <init>(): Quasiquotes.type = {
     Quasiquotes.super.<init>();
     ()
   };
   scala.meta.Importer.Initial.apply(
     scala.meta.Term.Select.Initial.apply(
       scala.meta.Term.Name.Initial.apply("scala"),
       scala.meta.Term.Name.Initial.apply("util")
     ),
     List.apply[meta.Importee.Name](
       scala.meta.Importee.Name.Initial.apply(
         scala.meta.Name.Indeterminate.Initial.apply("Try")
       )
     )
   )
 }
}
```

https://xuwei-k.github.io/scalameta-ast

```scala
import scala.util.Try
```

```scala
import scala.meta._
import scalafix.Patch
import scalafix.lint.Diagnostic
import scalafix.lint.LintSeverity
import scalafix.v1.SyntacticDocument
import scalafix.v1.SyntacticRule

class Example extends SyntacticRule("Example") {
  override def fix(implicit doc: SyntacticDocument): Patch = {
    doc.tree.collect {
      case t @ Import(
            List(
              Importer(
                Term.Select(Term.Name("scala"), Term.Name("util")),
                List(Importee.Name(Name("Try")))
              )
            )
          ) =>
        Patch.lint(
          Diagnostic(
            id = "",
            message = "",
            position = t.pos,
            explanation = "",
            severity = LintSeverity.Warning
          )
        )
    }.asPatch
  }
}
```

Accessing compiler info via SemanticDB

- `Tree.symbol`
  - Get information from the typechecker
- `Term.synthetics`
  - Get access to implicit params application or syntactic sugar added by the compiler
- `SemanticDocument.diagnostics`
  - Check contextual compiler warnings

Generating side effects

The Patch API

```scala
object Patch {
 def lint(msg: Diagnostic): Patch

 def removeImportee(importee: scala.meta.Importee): Patch
 def addGlobalImport(importer: scala.meta.Importer): Patch

 def removeToken(token: Token): Patch
 def replaceToken(token: scala.meta.Token, toReplace: String): Patch
 def addLeft(tok: Token, toAdd: String): Patch
 def addRight(tok: Token, toAdd: String): Patch

 def replaceTree(from: Tree, to: String): Patch
 def addLeft(tree: Tree, toAdd: String): Patch
 def addRight(tree: Tree, toAdd: String): Patch
}
```

```scala
sealed abstract class Patch
  extends Product {

  def +(other: Patch): Patch
  def isEmpty: Boolean
  def atomic: Patch
}
```

Iterating & deploying

Bootstrap a sbt project with giter8

```
$ sbt new scalacenter/scalafix.g8 --repo="Repository Name"


scalafix
├── rules/src/main
│   ├── resources/META-INF/services
│   │   └── scalafix.v1.Rule
│   └── scala/fix
│       └── Rewrite.scala
├── input/src/main/scala/test
│   └── RewriteTest.scala
├── output/src/main/scala/test
│   └── RewriteTest.scala
└── tests/src/test/scala/fix
    └── RuleSuite.scala
```

Expect tests via the testkit - input/output

```
/*
rule = NamedLiteralArguments
*/
package test

object NamedLiteralArguments {
 def complete(isSuccess: Boolean): Unit = ()
 def finish(n: Int, isError: Boolean): Unit = ()
 complete(true)
 complete(isSuccess = true)
 complete(false)
 complete(false) // scalafix:ok; rule suppression
 finish(2, true)
}
```

```
package test

object NamedLiteralArguments {
 def complete(isSuccess: Boolean): Unit = ()
 def finish(n: Int, isError: Boolean): Unit = ()
 complete(isSuccess = true)
 complete(isSuccess = true)
 complete(isSuccess = false)
 complete(false) // scalafix:ok; rule suppression
 finish(2, isError = true)
}
```

Deployment

- From source (file: / http: / github:)
  - Limited to rules defined in a single file with no dependency other than Scalafix/Scalameta
  - Source compatibility is attempted, but not guaranteed over time
- Publish to Maven / Nexus
  - Point `scalafixDependencies` to the right artifacts
- Local sbt project
  - Host them alongside the target code in your monolith
- Scala Steward
  - Open a PR for open source projects
  - Maintain your own configuration for your hosted Steward instance

👍 DOs

- Use quasi-quotes for prototyping
- Delegate formatting to scalafmt
- Check that your rule is idempotent
- Use `.atomic` on Patches to support suppressing diagnostics & rewrites
- Generate dialect-specific syntax with pretty-printing

🚫 DON'Ts

- Store state in a Rule instance
- Compare Scalameta trees with ==
- Patch trees or token larger than you need

Thank you!

More questions?

#scalafix on Scalameta discord

Code generation

```
@data class Person(name: String, age: Int)
```

```
@data final class Person private (val name: String, val age: Int) extends Product with Serializable {
  override def equals(obj: Any): Boolean = obj match {
    case c: Person =>
      this.name == c.name && this.age == c.age
    case _ =>
      false
  }
  override lazy val hashCode: Int = {
    val state = Seq(name, age)
    state.foldLeft(0)((a, b) => 31 * a.hashCode() + b.hashCode())
  }
...
```

Use-cases

diagnostics
- prevent usage of certain execution contexts
- enforce onion architecture

rewrites
- annotate public methods with LLM-generated comments

What’s missing/next? Contribute

rule users
ExplicitResultTypes in Scala 3

Rule authors
creating/renaming files

internals
Leverage TASTy instead of SemanticDB

Contribute https://github.com/scalacenter/scalafix/contribute