Macros

Scala and Metaprogramming

Arianna Soriani
arianna.soriani@studio.unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna

Academic Year 2023/2024

Outline

1. Metaprogramming
2. The 5W of Macros
3. Scala 2 Macros
4. Scala 3 Macros
5. Scala 2 vs Scala 3 Macros

1. Metaprogramming
2. The 5W of Macros
3. Scala 2 Macros
4. Scala 3 Macros
5. Scala 2 vs Scala 3 Macros

Metaprogramming I

Metaprogramming ['70-'80]

Metaprogramming is programming that manipulates programs *as data*.

- Takes a program as input, outputs another program

Example: take a program P, return a program Q that executes P and then prints how long the execution of P took

Metaprogramming II

Uses of metaprogramming

- Generating code to avoid boilerplate
- Analyzing code to detect errors
- Transforming code, e.g. into more performant code
- Running computation at compile-time instead of at runtime
- ...

Metaprogramming III

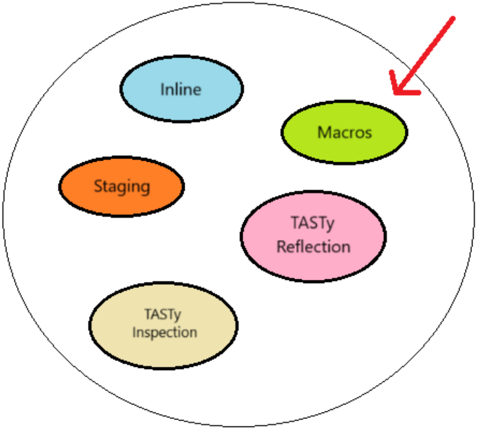


Figure: Fundamental features of metaprogramming - *Scala 3*

1. Metaprogramming
2. The 5W of Macros
3. Scala 2 Macros
4. Scala 3 Macros
5. Scala 2 vs Scala 3 Macros

What

Def.

Macros are metaprograms that treat programs as data

- which allows the user to analyze, manipulate and generate them at compile-time

With Macros it's possibile to write method that are executed at compile-time; these methods can generate codes that can be used normally

1. Metaprogramming
2. The 5W of Macros
   - Where
3. Scala 2 Macros
4. Scala 3 Macros
5. Scala 2 vs Scala 3 Macros

Where I

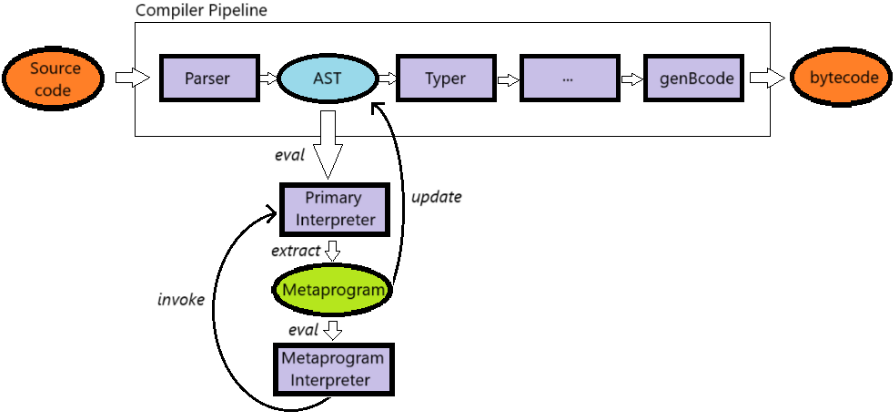


Figure: Compiler phases and Metaprogramming

Where II

Growth

Macros have become very useful tools in many contexts.
Widely used in:

- Several popular libraries
- In many industries
- As part of the research
- Many Scala constructs are based on the help of Macros
- ...

1 Metaprogramming

2 The 5W of Macros
- Who

3 Scala 2 Macros

4 Scala 3 Macros

5 Scala 2 vs Scala 3 Macros

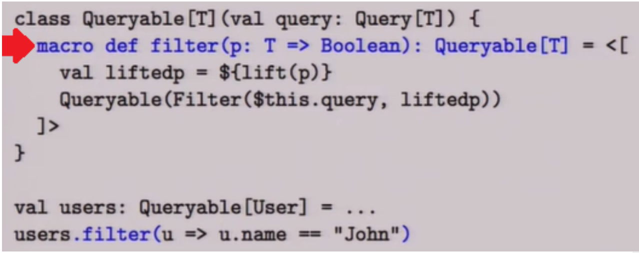
Who I

Eugene Burmako & Martin Odersky

- A significant part of the inspiration and early development of Scala Macros is attributed to *Eugene Burmako*.
- He is inspired by metaprogramming concepts present in other languages.
- The project was originally conceived as a fun experiment, but it evolved into a stable feature that was included in Scala 2.10.

Who II

- From **def-side** to call-side:



```
class Queryable[T](val query: Query[T]) {
  macro def filter(p: T => Boolean): Queryable[T] = <[
    val liftedp = ${lift(p)}
    Queryable(Filter($this.query, liftedp))
  ]>
}

val users: Queryable[User] = ...
users.filter(u => u.name == "John")
```

Figure: First Sketch of a Macro - *Scala 2*

1 Metaprogramming

2 The 5W of Macros
- When

3 Scala 2 Macros

4 Scala 3 Macros

5 Scala 2 vs Scala 3 Macros

When

A bit of history...

1. **2011-2012**: Initial research and development on Scala Macros.
2. **2013**: Release of Scala 2.10, which included the first version of Scala Macros, largely based on Burmako's work.
3. **Later Work**: He continued refining Macros and updated it in the later version of Scala.
4. **2021**: New macro system with the release of Scala 3.

1 Metaprogramming

2 The 5W of Macros
- Why

3 Scala 2 Macros

4 Scala 3 Macros

5 Scala 2 vs Scala 3 Macros

Why

Why this need?

- Because the demand from industries for the use of metaprogramming is increasing more and more.
- e.g. Slick project

1. Metaprogramming
2. The 5W of Macros
3. Scala 2 Macros
4. Scala 3 Macros
5. Scala 2 vs Scala 3 Macros

Idea of Scala Macro

Idea

"Just a single feature: unhygienic expansion of typed method calls" [Burmako, 2013]

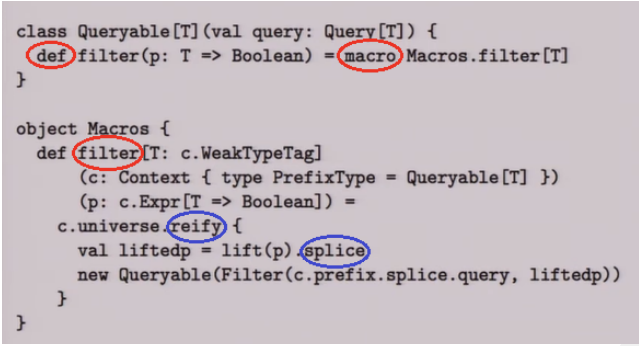
Native feature

- Normal Scala function: *def ... = macro {...}*
- Using reflection

Adding feature

- hygiene
- quasiquote

Example



```
class Queryable[T](val query: Query[T]) {
  def filter(p: T => Boolean) = macro Macros.filter[T]
}

object Macros {
  def filter[T: c.WeakTypeTag]
      (c: Context { type PrefixType = Queryable[T] })
      (p: c.Expr[T => Boolean]) =
    c.universe.reify {
      val liftedp = lift(p).splice
      new Queryable(Filter(c.prefix.splice.query, liftedp))
    }
}
```

Figure: e.g. Scala 2 Macro

Pro

Goal

Use for intelligent solutions for difficult architectural problems in some advanced libraries.

Advantages

1. The Macros have achieved great success in a short time.
2. Users who use Macro functions don't realize they exist!
3. Normal typed method calls.
4. Some Scala features are internally improved thanks to the use of Macros.

But...

A lot of disadvantages!

1. Scala 2 Macros architecture are finely coupled with the Scala 2 compiler architecture.
2. It becomes difficult to port them to the new compiler Dotty.
3. Complex API.
4. Always sperimental.
5. Advanced knowledge is required.

1. Metaprogramming
2. The 5W of Macros
3. Scala 2 Macros
4. Scala 3 Macros
5. Scala 2 vs Scala 3 Macros

The new macro system

Scala 3 Macro system

- All the advantages of Scala 2 Macros and much more.
- Completely redefined.
- Focus on security and robusteness.
- Compiler-independent API.
- API that scales in complexity.
- TASTy as a compatibility layer.
- Hygienic per default.
- Much much simpler!.
- Offers almost the same level of power, but in a more accessible way.

Migration

Recipe for rewriting new Macros

1. **Don't use a Macro!**
2. Use *inline*
3. Use *scala.quoted*
4. Directly modify ASTs and use *reflection*
5. Use a plugin

1 Metaprogramming

2 The 5W of Macros

3 Scala 2 Macros

4 Scala 3 Macros
- Inline

5 Scala 2 vs Scala 3 Macros

1) Inline I

Def.

When a method is declared as *inline*, the compiler replaces all its invocations in the source code with its body, as if it were "copied" in place of the call.

But also...

- Inline parameters
- Transparent inline
- Inline conditionals
- etc.

1) Inline II

Advantages

- Performance optimization.
- More effective conditional expressions.
- Integration with metaprogramming.

But... be careful!

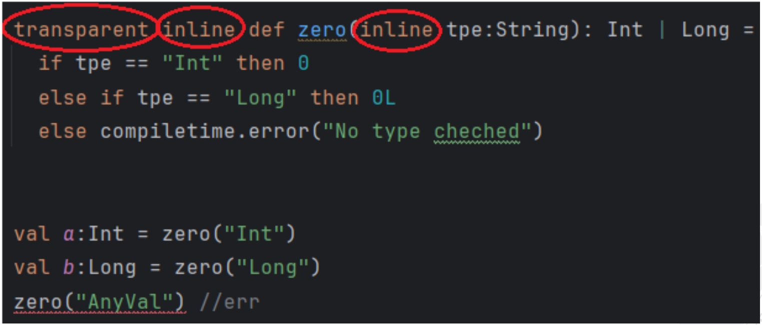
- Don't abuse it! Exponential increase in code size.

Example

Transparent inline

Inline method

Inline parameter

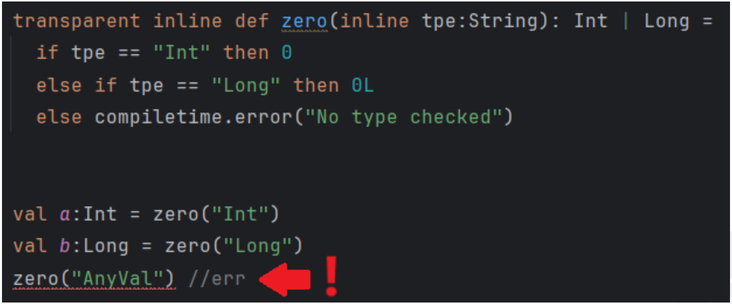


```
transparent inline def zero(inline tpe:String): Int | Long =
  if tpe == "Int" then 0
  else if tpe == "Long" then 0L
  else compiletime.error("No type cheched")


val a:Int = zero("Int")
val b:Long = zero("Long")
zero("AnyVal") //err
```

Figure: e.g. Inline

scala.compiletime



```
transparent inline def zero(inline tpe:String): Int | Long =
  if tpe == "Int" then 0
  else if tpe == "Long" then 0L
  else compiletime.error("No type checked")


val a:Int = zero("Int")
val b:Long = zero("Long")
zero("AnyVal") //err
```

Figure: e.g. scala.compiletime

1. Metaprogramming

2. The 5W of Macros

3. Scala 2 Macros

4. Scala 3 Macros
   - scala.quoted

5. Scala 2 vs Scala 3 Macros

2) scala.quoted

Quoting '{...}

It is the process of transforming a block of code into an object of type Expr[T], which represents an typed expression at the AST level.

- Allows you to manipulate code *as data* at compile-time.

Splicing ${...}

It is the reverse process compared to quoting; allows the insertion of an Expr[T] inside another quote.

Quotation e Splicing

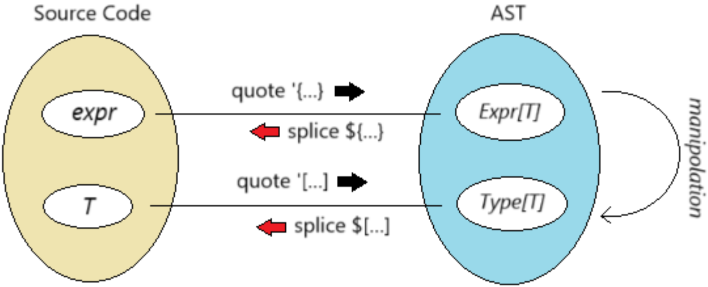
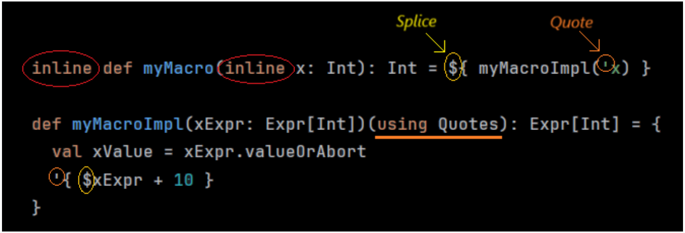


Figure: Quotation e Splicing

Complete Macro

Macro = Inline + Quote + Splice



```scala
inline def myMacro(inline x: Int): Int = ${ myMacroImpl('x) }

def myMacroImpl(xExpr: Expr[Int])(using Quotes): Expr[Int] = {
  val xValue = xExpr.valueOrAbort
  '{ $xExpr + 10 }
}
```

Figure: e.g. Scala 3 Macro

Quote

Quote

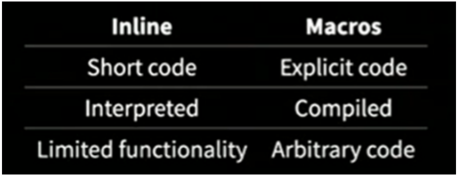
- Context Object in which all macro operations are carried out ("entry point").
- Every macro implementation must *always* have an instance of **scala.reflect.Quotes** available.

Purposes

1. Defines the type scope of the AST.
2. Captures the expansion context of the Macro.

1. Metaprogramming
2. The 5W of Macros
3. Scala 2 Macros
4. Scala 3 Macros
   - Inline vs Macros
5. Scala 2 vs Scala 3 Macros

Inline vs Macros



Inline	Macros
Short code	Explicit code
Interpreted	Compiled
Limited functionality	Arbitrary code

Figure: Inline vs Macros

Macros are ...

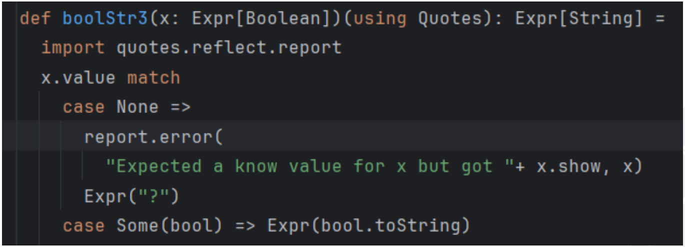
- Faster
- More flexible
- More powerful
- But.. more complex!

1. Metaprogramming
2. The 5W of Macros
3. Scala 2 Macros
4. Scala 3 Macros
   - Reflection
5. Scala 2 vs Scala 3 Macros

3) quotes.reflect._ I

Use of Reflection

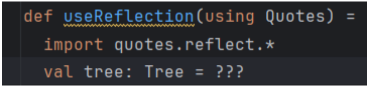
- Error reporting
- Visualization, creation and manipulation of AST-based code.
- Provides information about the files that have been compiled.



```scala
def boolStr3(x: Expr[Boolean])(using Quotes): Expr[String] =
  import quotes.reflect.report
  x.value match
    case None =>
      report.error(
        "Expected a know value for x but got "+ x.show, x)
      Expr("?")
    case Some(bool) => Expr(bool.toString)
```

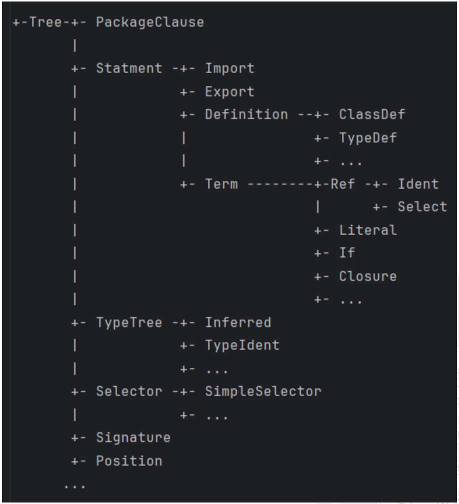
Figure: e.g. Error Reporting

3) quotes.reflect._ II



```scala
def useReflection(using Quotes) =
  import quotes.reflect.*
  val tree: Tree = ???
```

(a) e.g. AST Manipolation



```
+-Tree-+- PackageClause
       |
       +- Statment -+- Import
       |            +- Export
       |            +- Definition --+- ClassDef
       |            |               +- TypeDef
       |            |               +- ...
       |            +- Term --------+-Ref -+- Ident
       |                            |      +- Select
       |                            +- Literal
       |                            +- If
       |                            +- Closure
       |                            +- ...
       +- TypeTree -+- Inferred
       |            +- TypeIdent
       |            +- ...
       +- Selector -+- SimpleSelector
       |            +- ...
       +- Signature
       +- Position
      ...
```

(b) Structure of AST

1 Metaprogramming

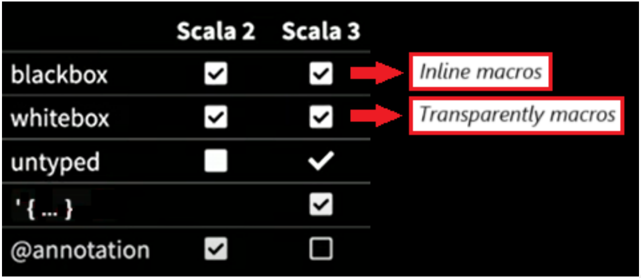
2 The 5W of Macros

3 Scala 2 Macros

4 Scala 3 Macros

5 Scala 2 vs Scala 3 Macros

Scala 2 *vs* Scala 3 Macros I



	Scala 2	Scala 3	
blackbox	☑	☑	→ *Inline macros*
whitebox	☑	☑	→ *Transparently macros*
untyped	□	✓	
'{ ... }		☑	
@annotation	☑	□	

Figure: Features comparison

Scala 2 *vs* Scala 3 Macros II

Benefit of the new Scala Macro System

- Simpler
- It can be within the reach of even the least experienced (for simple applications)
- Completely transparent to the user
- No longer experimental
- Also uses simple native scale constructs
- Safer and more robust
- ...
- Completely renovated!

Macros

Scala and Metaprogramming

Arianna Soriani
arianna.soriani@studio.unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna

Academic Year 2023/2024

References

[Burmako, 2013] Burmako, E. (2013).
Scala macros: Let our powers combine!
.

[Horstmann, 2022] Horstmann, C. S. (2022).
Scala for the impatient, 3rd edition
.

[Odersky and Moors, 2018] Odersky, M. and Moors, A. (2018).
Preparing for scala 3
.

[Wampler, 2021] Wampler, D. (2021).
Programming scala, scalability = functional programming + objects, 3rd edition
.