Modular Quasiquotes for Scala

Denys Shabalin

École Polytechnique Fédérale de Lausanne

2 July 2013

Q: What are quasiquotes?

Q: What are quasiquotes?

A: A composable syntactical abstraction that vastly simplifies manipulation of ASTs.

Compactness

Syntax	`case class Foo(bar: Baz)`

Compactness

Syntax	`case class Foo(bar: Baz)`
AST	`ClassDef(Modifiers(...), TypeName("Foo"), List(), Template(List(Select(Ident(TermName("scala")), TypeName("Product")), Select(Ident(TermName("scala")), TypeName("Serializable"))), emptyValDef, List(ValDef(Modifiers(...), TermName("bar"), Ident(TypeName("Baz")), EmptyTree), DefDef(Modifiers(), nme.CONSTRUCTOR, List(), List(List(ValDef(Modifiers(...), TermName("bar"), Ident(TypeName("Baz")), EmptyTree))), TypeTree(), Block(List(pendingSuperCall), Literal(Constant(()))))))))`

Compactness

Syntax	`case class Foo(bar: Baz)`
AST	`ClassDef(Modifiers(...), TypeName("Foo"), List(), Template(List(Select(Ident(TermName("scala")), TypeName("Product")), Select(Ident(TermName("scala")), TypeName("Serializable"))), emptyValDef, List(ValDef(Modifiers(...), TermName("bar"), Ident(TypeName("Baz")), EmptyTree), DefDef(Modifiers(), nme.CONSTRUCTOR, List(), List(List(ValDef(Modifiers(...), TermName("bar"), Ident(TypeName("Baz")), EmptyTree))), TypeTree(), Block(List(pendingSuperCall), Literal(Constant(())))))`
Quasiquote	`q"case class Foo(bar:  Baz)"`

Composability

```
// it's easy to combine quasiquotes

val tree = q"simple tree"
val another = q"if ($tree) foo else bar"
```

Composability

```
// it's easy to combine quasiquotes

val tree = q"simple tree"
val another = q"if ($tree) foo else bar"

// and they can also be used to
// decompose trees in the same fashion

tree match {
  case q"$obj.$member" =>
    // obj & member are now available in this scope
}
```

Expressiveness

```
// you can splice lists of elements into a tree with
// the help of special cardinality annotation

val args = List(q"a", q"b")
q"f(..$args)"
```

Expressiveness

```
// you can splice lists of elements into a tree with
// the help of special cardinality annotation

val args = List(q"a", q"b")
q"f(..$args)"

// equivalent to

q"f(a, b)"
```

Expressiveness

```
// you can splice lists of elements into a tree with
// the help of special cardinality annotation

val args = List(q"a", q"b")
q"f(..$args)"

// equivalent to

q"f(a, b)"

// and non-tree data types

val i = 0
q"f($i)"
```

Modularity

```
@quasiquote object λ {

}
```

Modularity

```
@quasiquote object λ {
  sealed abstract class Tree
  case class Abs(v: Var, body: Tree) extends Tree
  case class App(f: Tree, arg: Tree) extends Tree
  case class Var(name: String) extends Tree









}
```

Modularity

```
@quasiquote object λ {
  sealed abstract class Tree
  case class Abs(v: Var, body: Tree) extends Tree
  case class App(f: Tree, arg: Tree) extends Tree
  case class Var(name: String) extends Tree

  object parse extends StdTokenParsers {









  }
}
```

Modularity

```
@quasiquote object λ {
  sealed abstract class Tree
  case class Abs(v: Var, body: Tree) extends Tree
  case class App(f: Tree, arg: Tree) extends Tree
  case class Var(name: String) extends Tree

  object parse extends StdTokenParsers {
    lexical.delimiters ++= List("(", ")", "\", ".")
    def main   = rep1(parens | varr | abs | hole)        ^^ App
    def abs    = ("\\" ~> (varr | hole) <~ ".") ~ main) ^^ Abs
    def varr   = ident                                   ^^ Var
    def parens = "(" ~> main <~ ")"
  }
}
```

Modularity

```
// now we can use our custom quasiquotes to construct

import λ._

val id = λ"\x. x"
val f = λ"\v. $id v"
```

Modularity

```
// now we can use our custom quasiquotes to construct

import λ._

val id = λ"\x. x"
val f = λ"\v. $id v"

// and deconstruct our lambda-calculus trees

f match {
  case λ"\$arg. $body" =>
}
```

Summary

- Quasiquotes are an extremely powerful abstraction over ASTs
- Primary usage is to simplify manipulation of Scala trees
- However they can be generalized to arbitrary languages
- Our framework derives implementations from declarative definitions