Let's Unify With Scala Pattern Matching!

Edmund Lam
sllam@qatar.cmu.edu

Iliano Cervesato
iliano@cmu.edu

Carnegie Mellon University, Qatar Campus

Supported by QNRF grant JSREP 4-003-2-001 and NPRP 4-341-1-059

26 June 2016

Scala, a Modern, General-Purpose, Programming Language

- Tons of modern features:
  - Object-oriented programming
  - Functional programming
  - Algebraic data types
  - Extensible pattern matching
  - Type inference
  - Lazy evaluation
  - The list goes on . . .

Scala, a Modern, General-Purpose, Programming Language

- Tons of modern features:
  - Object-oriented programming
  - Functional programming
  - Algebraic data types
  - Extensible pattern matching
  - Type inference
  - Lazy evaluation
  - The list goes on . . .
- But . . . no love for unification . . . =(
  - No built-in support
  - No official libraries

Algebraic Data Types and Pattern Matching in Scala

- Defining algebraic data types:

```
abstract class Term
case class Var(name: String) extends Term
case class Fun(arg: String, body: Term) extends Term
case class App(f: Term, v: Term) extends Term
```

- Built-in support for pattern matching:

```
def printTerm(term: Term) {
    term match {
        case Var(n)    => print(n)
        case Fun(x, b) => print("^" + x + ".")
                          printTerm(b)
        case App(f, v) => printTerm(f)
                          print(" ")
                          printTerm(v)
    }
}
```

Unification Library in Scala

- Wouldn't it be great if Scala provided control statements for *unification*?
  - Similar to match statement for pattern matching

Unification Library in Scala

- Wouldn't it be great if Scala provided control statements for *unification*?
  - Similar to match statement for pattern matching
- That's what our Scala unification library does!

```
val x: Term = new LogVar()
val y: Term = new LogVar()
val f: Term = F(Const(5),x)
f unify (
   Const(4) withMgu θ => {
      ...

   },
   F(y,Const(4)) >=> {
      ...
   }
)
```

Unification Library in Scala

- Wouldn't it be great if Scala provided control statements for *unification*?
  - Similar to match statement for pattern matching
- That's what our Scala unification library does!

```
val x: Term = new LogVar()    // declaring new logical variable x
val y: Term = new LogVar()
val f: Term = F(Const(5),x)
f unify (
   Const(4) withMgu θ => {
      ...

   },
   F(y,Const(4)) >=> {
      ...
   }
)
```

Unification Library in Scala

- Wouldn't it be great if Scala provided control statements for *unification*?
  - Similar to match statement for pattern matching
- That's what our Scala unification library does!

```
val x: Term = new LogVar()    // declaring new logical variable x
val y: Term = new LogVar()    // declaring new logical variable y
val f: Term = F(Const(5),x)
f unify (
   Const(4) withMgu θ => {
      ...

   },
   F(y,Const(4)) >=> {
      ...
   }
)
```

Unification Library in Scala

- Wouldn't it be great if Scala provided control statements for *unification*?
    - Similar to match statement for pattern matching
- That's what our Scala unification library does!

```
val x: Term = new LogVar()    // declaring new logical variable x
val y: Term = new LogVar()    // declaring new logical variable y
val f: Term = F(Const(5),x)   // f is the term F(Const(5),x)
f unify (
   Const(4) withMgu θ => {
      ...

   },
   F(y,Const(4)) >=> {
      ...
   }
)
```

Unification Library in Scala

- Wouldn't it be great if Scala provided control statements for *unification*?
  - Similar to match statement for pattern matching
- That's what our Scala unification library does!

```
val x: Term = new LogVar()    // declaring new logical variable x
val y: Term = new LogVar()    // declaring new logical variable y
val f: Term = F(Const(5),x)   // f is the term F(Const(5),x)
f unify (                     // the unification control statement
   Const(4) withMgu θ => {
      ...

   },
   F(y,Const(4)) >=> {
      ...
   }
)
```

Unification Library in Scala

- Wouldn't it be great if Scala provided control statements for *unification*?
  - Similar to match statement for pattern matching
- That's what our Scala unification library does!

```
val x: Term = new LogVar()     // declaring new logical variable x
val y: Term = new LogVar()     // declaring new logical variable y
val f: Term = F(Const(5),x)    // f is the term F(Const(5),x)
f unify (                      // the unification control statement
   Const(4) withMgu θ => {     // try unifying f and Const(4), producing mgu θ
      ...                      //   it's pure: no side-effects on x and y,
                               //              substitution θ available
   },
   F(y,Const(4)) >=> {
      ...
   }
)
```

Unification Library in Scala

- Wouldn't it be great if Scala provided control statements for *unification*?
  - Similar to match statement for pattern matching
- That's what our Scala unification library does!

```
val x: Term = new LogVar()     // declaring new logical variable x
val y: Term = new LogVar()     // declaring new logical variable y
val f: Term = F(Const(5),x)    // f is the term F(Const(5),x)
f unify (                      // the unification control statement
   Const(4) withMgu θ => {     // try unifying f and Const(4), producing mgu θ
      ...                      //   it's pure: no side-effects on x and y,
                               //              substitution θ available
   },
   F(y,Const(4)) >=> {         // try unifying f and F(y,Const(4)), ''imperatively''
      ...                      //   mgu [5/y, 4/x] applied to x and y as side-effect
   }
)
```

Unification with Extensible Pattern Matching

- An alternative abstraction
- Our unification library "integrates" with pattern matching:

```
val x: Term = new LogVar()
val y: Term = new LogVar()
val f: Term = F(Const(5),x)
val unifA = new Unif( Const(4) )          // ''Unification extractor'' for Const(4)
val unifB = new Unif( F(y,Const(4)) )     // ''Unification extractor'' for F(y,Const(4))
f match {
    case unifA(θ) => ...                  // Try unifying with Const(4) and extract θ
    case unifB(θ) => ...                  // Try unifying with F(y,Const(4)) and extract θ
}
```

Unification with Extensible Pattern Matching

- An alternative abstraction
- Our unification library "integrates" with pattern matching:

```
val x: Term = new LogVar()
val y: Term = new LogVar()
val f: Term = F(Const(5),x)
val unifA = new Unif( Const(4) )          // ''Unification extractor'' for Const(4)
val unifB = new Unif( F(y,Const(4)) )     // ''Unification extractor'' for F(y,Const(4))
f match {
    case unifA(θ) => ...                  // Try unifying with Const(4) and extract θ
    case unifB(θ) => ...                  // Try unifying with F(y,Const(4)) and extract θ
}
```

- It's possible, with *extensible pattern matching*!

Extensible Pattern Matching in Scala

- User-definable *pattern extractors*, to be used in Scala's match statements
- A classic example:

```
object Twice {
  def unapply(x: Int): Option[Int] = if(x%2==0) Some(x/2) else None
  def test(x: Int) {
     x match {
        case Twice(y) => println(x + "is even and twice " + y)
        case _ => println(x + " is odd")
     }
  }
}
```

- to obtain `y`, `unapply` is implicitly called in the match statement

Extensible Pattern Matching in Scala

- User-definable *pattern extractors*, to be used in Scala's match statements
- A classic example:

```
object Twice {
  def unapply(x: Int): Option[Int] = if(x%2==0) Some(x/2) else None
  def test(x: Int) {
     x match {
        case Twice(y) => println(x + "is even and twice " + y)
        case _ => println(x + " is odd")
     }
  }
}
```

- to obtain `y`, `unapply` is implicitly called in the match statement
- Unification extractor `Unif` defines a "family" of unification pattern extractors:

```
class Unif[A](pat: Term[A]) {
    def unapply(t: Term[A]): Option[Subst] =
       t.mgu(pat)     // return mgu of t and pat if it exists (option type)
}
```

Current Status

- Open-source and available at:

  `https://github.com/sllam/unifscala`

- Please star it!
- Future works:
  - Higher level combinators (e.g., backtracking, constraint solving)
  - Unification over sets and multisets

Thank you!

Questions please?