Yin-Yang: Transparent Deep Embedding of DSLs

Vojin Jovanovic, EPFL
Twitter: @vojjov
Good Performance: SQL

1. Compiler has domain knowledge

2. Compiled at run-time (access to data)
Deep Embedding
Deep Embedding - LMS

```scala
trait Base {
  trait Exp[T]
  type Rep[T] = Exp[T]
  case class Const[T](t: T) extends Exp[T]
  implicit def unit[T](t: T): Rep[T] = Const(t)
}

trait RegexDSL extends Base {
  case class Matches(
    t: Rep[String],
    p: Rep[String]) extends Exp[Boolean]
  object regex {
    def matches(t: Rep[String], p: Rep[String]) = Matches(t, p)
  }
  def main() =
    regex.matches("42", "Answer to the Ultimate...")
}
```
Program Text is Not All

```python
regex.matches("42", "Answer to the Ultimate...")
```
Convoluted Interface

```python
def infix_-(lhs: Rep[Float], rh: Rep[Int])
    (implicit o: Overloaded,
     ctx: SourceContext): Rep[Float]
```

```python
def infix_-(lhs: Rep[Int], rh: Rep[Int])
    (implicit o: Overloaded,
     ctx: SourceContext): Rep[Int]
```
Type Errors

```scala
val one: Rep[Int] = 1
val void: Rep[Unit] = ()
one + void

No implicit view available from RepDSL.this.Rep[Unit] => Int.
```
Deep DSL Embedding

- Nice interface
- Comprehensible type errors
- Easy debugging
- Consistent Documentation
- Consistent with the host language

- Domain-specific analysis
- Fast
Shallow Embedding

package object regex {
  def regexDSL[T](b: => T) = b
  def matches(text: String, pat: String): Boolean = text.matches(pat)
}
Shallow Embedding

✓ Nice interface
✓ Comprehensible type errors
✓ Easy debugging
✓ Consistent documentation
✓ Consistent with the host language

X Domain-specific analysis
X Fast
During program development we do not care about performance!
✓ Use shallow embedding for development
✓ Use deep embedding in production
Macros

Compile-time meta-programming
Completely transparent to the users

```python
def fix_==[T](block: => T): T =
    macro fix_==Impl
```
Regular Workflow

```
... foo{"Bar" == 1}
...
...
foo{"Bar" == 1} // typed
...
```

Type Checker
Macro Workflow

... fix_=={"Bar" == 1} ...

fix_==Impl(Tree({"Bar" == 1}))

... _==("Bar", 1) // typed ...

Type Checker
Yin-Yang Library

Uses macros to reliably translate shallow programs to deep programs!
val readHGTG = ...; val text = "42";
val pattern = "Answer to the Ultimate Q..."
regexDSL {
    val res = if (readHGTG)
        matches(
            text,
            pattern
        )
    else true
    res
}
Ascription Transformation

val readHGTG = ...; val text = "42";
val pattern = "Answer to the Ultimate Q..."

regexDSL {
    val res: Boolean = ((if (readHGTG)
        (regex.`package`.matches(
            text,
            pattern
        ): Boolean)
    else true): Boolean)

    res
}

Lift Literals Transformation

val readHGTG = ...; val text = "42";
val pattern = "Answer to the Ultimate Q..."
regexDSL {
    val res: Boolean = ((if (readHGTG)
        (regex.`package`.matches(
            text,
            pattern
        ): Boolean)
    else lift(true)): Boolean)
    res
}

Virtualization Transformation

val readHGTG = ...; val text = "42";
val pattern = "Answer to the Ultimate Q..."
regexDSL {
    val res: Boolean = ((__ifThenElse(readHGTG,
        (regex.`package`.matches(
            text,
            pattern
        ): Boolean),
        lift(true)): Boolean)
    res
}
Scope Injection Transformation

val readHGTG = ...; val text = "42";
val pattern = "Answer to the Ultimate Q..."
regexDSL {
    val res: Boolean = ((__ifThenElse(readHGTG,
    (this.regex.`package`.matches(
    text,
    pattern
    ): Boolean),
    lift(true)): Boolean)
    res
}
Type Transformation

```scala
val readHGTG = ...; val text = "42"
val pattern = "Answer to the Ultimate Q..."
regexDSL {
  val res: this.Rep[Boolean] =
  ((__ifThenElse(readHGTG,
    (this.regex.`package`.matches(
      text,
      pattern
    ): this.Rep[Boolean]),
    lift(true)): this.Rep[Boolean])
  res
}
```
Hole Transformation

val readHGTG = ...; val text = "42";
val pattern = "Answer to the Ultimate Q..."
regexDSL {
  val res: this.Rep[Boolean] =
    ((__ifThenElse(hole(typeTag[Boolean], 1),
      (this.regex.\`package\`.matches(
        hole(typeTag[String], 2),
        hole(typeTag[String], 3)
      ): this.Rep[Boolean]),
      lift(true)): this.Rep[Boolean])
  res
}
val readHGTG = ...; val text = "42";
val pattern = "Answer to the Ultimate Q..."

new RegexDSL { def main() {
    val res: this.Rep[Boolean] =
        ((__ifThenElse(hole(typeTag[Boolean],1)
            (this.regex.`package`.matches(
                hole(typeTag[String], 2),
                hole(typeTag[String], 3)
            ): this.Rep[Boolean]),
            lift(true)): this.Rep[Boolean])
    res
}}
Reflective Instantiation

val dsl =
  c.eval(new RegexDSL {def main()={...}})
Domain-Specific Analysis

dsl.staticallyAnalyze(c)

Reports errors at compile time!
Feature Analysis

Prototyping?

Transformation to Deep Embedding

Static Analysis

Captured Identifiers Analysis

Interpretation or Run-Time Code Generation

Missing Features Error Reporting

Shallow Embedding

Deep Embedding

Domain-Specific Error Reporting

Compile-Time Code Generation

Instantiated DSL

Collected Identifiers

Interpreted or Staged?
Captured Identifiers Analysis

val requiredIdents = dsl.stagingAnalysis()
if (requiredIdents != Nil)

val readHGTG = ...; val text = "42"
val pattern = "Answer to the Ultimate Q..."
new RegexDSL { def main() {
  val res: this.Rep[Boolean] =
    ((__ifThenElse(hole(typeTag[Boolean],1)
      (this.regex.`package`.matches(
        hole(typeTag[String], 2),
        lift(pattern)
      ): this.Rep[Boolean]),
      lift(true)): this.Rep[Boolean])
  res
}
}
Compile vs. Runtime

```ruby
if (requiredIdents == Nil)
  // compile at compile time
  c.parse(dsl.generateCode())
else
  // compile at run time
  c.expr(Block(
    guards,
    dslCake,
    dslInvocation
  ))
```

Deep DLSs: Idents vs. Constants

• Deep embedding does not distinguish constants and identifiers

• To check for recompilation it needs to lift the whole program on each execution
Guards with Deep DSLS

```scala
val s = text.map(incChar)
if (matches(s, pattern))
  println("OK")
```

How long does the lifting take?

Shallow program processes 100 KB string for the time of one lifting!
Guarded Recompilation

```java
if (pattern != Cache.prevValue(<uid>))
    Cache.setProgram(<uid>)(
        new RegexDSL{
            def main()={…}}
    )

Cache.program(<uid>)(cond, text)
```
Evaluation

Guard Execution Time (ms)

P. Rank  Cond.  SCC  500 LoC  1000 LoC

OptiGraph Application

Delite Opt.  Yin-Yang
Contributions

• Completely transparent deep embedding

• Completely compiler agnostic

• Compilation at either compile or run time

• Efficient guarded recompilation
Slick DSL with Macors

- Macro version took months to develop
- Duplicate of the deep embedding
- Does not work for all cases
Macro Version of Slick

• Requires same things as Yin-Yang
  – Hole Transformation
  – Virtualization
  – Compile-time evaluation

• These transformation are non-trivial
Slick with Yin-Yang

• Three weeks development

• Wires to the existing DSL (no duplication)

• More features than the macro version
Future Work

• Class virtualization

• Cross compilation unit operation

• Yin-Yang as a modular library for DSLs
References

• Yin-Yang
  – http://github.com/vjovanov/mpde

• Learn LMS
  – http://scala-lms.github.com
Questions?

Also at:

@vojjov
vojin.jovanovic@epfl.ch