Scalan: a reasonably typed meta-programming framework in Scala

Alexander Slesarenko
Let us know what you think

Please use the Scala Days app to rate sessions.
Meta-programming

Why meta-programming?
- avoid overhead of abstractions
- automatic code optimizations
- generate boilerplate
- develop DSLs
- tweak syntax
- etc.

Meta-programming is typically hard
Why meta-programming is hard?

- Intermediate Representation (IR) is typically defined by Abstract Syntax.
- The data has complex recursive structure.
- All operations also have to be recursive.

- $\mathcal{T} \ni \tau ::= \text{Unit} \mid \text{Int} \mid (\tau_1 \times \tau_2) \mid (\tau_1 + \tau_2) \mid (\tau_1 \to \tau_2) \mid \text{Array}[\tau]$

$\text{Term} \ni e ::=  l \mid x : \tau \mid \lambda(x : \tau).e \parallel e_1 e_2 \mid \delta e \mid \text{case } e \text{ of } \{ p_i \to e_i \}$

$k ::= () \mid (-,-) \mid l[\tau_1, \tau_2] \cdot - \mid r[\tau_1, \tau_2] \cdot -$

$\delta ::= \text{fst}_- \mid \text{snd}_- \mid _- \oplus _-$

$p ::= l \mid k \ x \mid \lambda(x : \tau).e$

$v ::= l \mid k \ v \mid \lambda(x : \tau).e$

The hard part is to manipulate such data structures in semantically correct way.
How Scalan can help?

- **DAG based intermediate representation (IR)**
  - unified and extensible data structure
  - immutable and higher-order (HOAS)
  - configurable visualization (Graphviz)

- **Composable and generic representation for functions (lambdas) as DAGs**
- **virtualized user-defined types (UDTs)**
- **domain-specific isomorphisms**
- **domain-specific converters**

- **Composable and generic meta-operators (later in slides)**
Basic Machinery: Code Virtualization

1) Process an abstract syntax tree of a source code
2) Associate nodes of the AST with calls of a virtualized API
3) Generate a virtualized code, which contain only the calls of the virtualized API.

```scala
def mvm(matr: Matr[T], vec: Vec[T]): Vec[T] = 
  new DenseVec(
    matr.rows.map(
      { r: Vec[T] => r.dot(vec) }
    ))
```

```scala
  DenseVec.apply(
    matr.rows.map(
      fun { r: Rep[Vec[T]] => r.dot(vec) }
    ))
```
Basic Machinery: Standard Evaluation

Standard evaluation

Redexes

$$\text{new Array}(3).\text{size}$$

$$\text{new Array}(3).\text{size}$$

$$\text{Array(...).size}$$

3
Basic Machinery: Staged evaluation

- Staged evaluation can be understood as a *self-reproducing process*. When a program is stage-evaluated it reproduces itself in a graph-based IR.

```java
{ 
  val s34 = 3
  val s35 = new Array(s34)
  val s36 = s35.size
}

rule(\forall n =>
  new Array(n).size → n)

{ 
  val s34 = 3
  s34
}
Basic Machinery: Representation of Lambdas

```scala
trait AstGraph {
  val boundVars: List[Sym[_]]
  val roots: List[Sym[_]]
  lazy val freeVars: Set[Sym[_]]
  lazy val schedule: Seq[TableEntry[_]]
}

class Lambda[A, B](
  val f: Option[Sym[A] => Sym[B]]
  val x: Sym[A])(val y: Sym[B] = f(x),
  val mayInline: Boolean)
  extends BaseDef[A => B] with AstGraph {
    val boundVars = List(x)
    val roots = List(y)
    ...
  }
```
Semantics of the Virtualized Code

Virtualized code is semantically connected to the source code in two ways:

1) Code Virtualization produces $P_{VIRT}$ which is equivalent to Semantic Function with respect to Std. Evaluation

2) Staged Evaluation produces graph-based IR $P_{IR}$, which is equivalent to the Std. Evaluation with respect to Compilation & Run

While $P_{VIRT}$ can be executed directly (as JVM bytecode) Compiler is required to generate an executable machine code from $P_{IR}$
DEMONSTRATION
(1)
Basic Machinery: Generic meta-operators

- Isomorphisms
Basic Machinery: Generic meta-operator

- Converters

```python
class SparseMatrix

class DenseMatrix
```

```
getConverter
```

sparse2dense

```
0 1 2 3
0 1.0 0 2.0 0
1 3.0 4.0 5.0 0
2 0 0 0 6.0
```
Basic Machinery: Generic meta-operators

- Isomorphic Specialization

\[ f' = SE[to >> f >> from] \]
DEMONSTRATION
(2)
Converters for free

```scala
val Some(converter) = getConverter(
  element[((SparseMatrix[Double], Vector[Double])],
  element[((DenseMatrix[Double], Vector[Double)])]
)

lazy val mvm = fun {
  p: Rep[(Matrix[Double], Vector[Double])] =>
  val Pair(m, v) = p
  DenseVector((m.rows.map { r => r dot v })
)
```

(SparseMatrix, Vector)  

(DenseMatrix, Vector)

converter  

converter  >>

lazy val mvm = fun {
  p: Rep[(Matrix[Double], Vector[Double])] =>
  val Pair(m, v) = p
  DenseVector(m.rows.map { r => r dot v })
}
DEMONSTRATION

(3)
Generic meta-operators

- **Isomorphisms**
- **Converters**

**Isomorphic Specialization**

\[ A \approx IS \approx B \]

\[ A' \approx f' \approx B' \]
First-class Rewriting Rules

Scalan Context

- Scalac compiler works as a provider of the pattern and replacement graphs (Rewrite rule components)
- It performs a staged evaluation of the set of rules and creates the IRs
- Rewrite rule component keeps for each pattern graph a corresponding replacement graph

### Rule Specification

```
val mapFusion: Rewrite[A] = postulate(
  (xs: Rep[Array[A]], f: Rep[A => B], g: Rep[B => C]) =>
  xs.map(f).map(g) => xs.map((x: Rep[A]) => g(f(x)))
)
```

### Register for rewriting

```
val mapFusion[A,B,C]: RRewrite[Arr[C]] = postulate(
  (xs: Rep[Array[A]], f: Rep[A => B], g: Rep[B => C]) =>
  xs.map(f).map(g) => xs.map((x: Rep[A]) => g(f(x)))
)
```
Multi-stage Compilation Pipeline

Step 0: initial graph creation

Steps 1…n: graph transformations (mirroring + rewriting)

Step n+1: emit kernel code based on the final graph

Step n+2: compile kernels

C++  Scala  Lua

Step n+3: execute
Get involved

• Read the post about Scalan’s Meta-programming Idioms (https://github.com/scalan/scalan.github.io/blob/master/idioms.md)
• Check out the project at https://github.com/scalan/scalan
• Ask questions on Google Group https://groups.google.com/d/forum/scalan
• Follow us on twitter (@avslesarenko, @alexey_r)
• Code for the demo: https://github.com/scalan/scalan-starter
• Program Functionally, Execute Imperatively: Peeling abstraction overhead from functional programs. Scala Days Amsterdam, June 2015 (slides/video)
• Scalan publications: http://pat.keldysh.ru/~slesarenko/

Thank you
Please

Remember to rate this session

Thank you!
SCALA DAYS
BERLIN 2016