

Implementing a Universe Type System for Scala

Manfred Stock

ETH Zurich

Master Thesis Interim Presentation
06.12.2007

Introduction

Scala

- ▶ Functional
 - ▶ Every function is a value
- ▶ Object oriented
 - ▶ Every value is an object
 - ▶ Every operation is a message send
- ▶ Interoperable with Java
 - ▶ Compiled to Java bytecode
 - ▶ Access to Java classes

Universe Type System

- ▶ Ownership topology
- ▶ Owner-as-modifier for encapsulation

Outline

Introduction
Outline
Scala
Scala Compiler Plugins

Architecture
Universe Plugins

Implementation
Inference

Conclusion
Problems
Status and Next Steps

Scala

Most relevant features

- ▶ Type inference for variables, method return types and type parameters
- ▶ Type annotations
- ▶ Plugins for the compiler

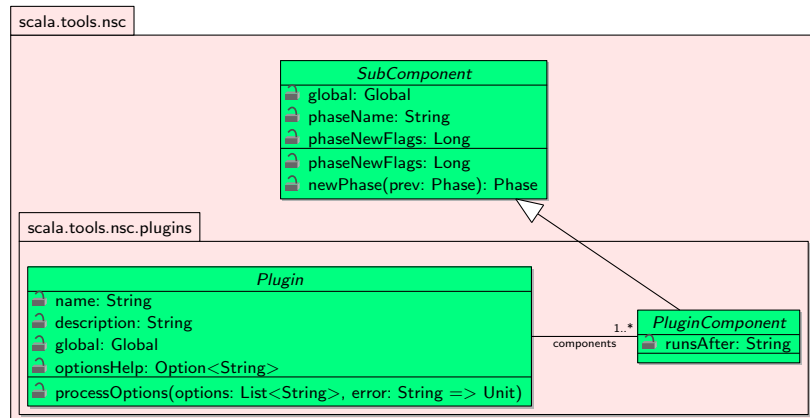
Exemplary use of annotations

```

0 package ch.ethz.inf.sct.uts.examples
  import ch.ethz.inf.sct.uts.annotation._
2 class A {
    val b = new (B @rep)
4    val s = b.s
    def t = new (Cls @peer)
6    val u = b.t
  }

```

Scala Compiler Plugins



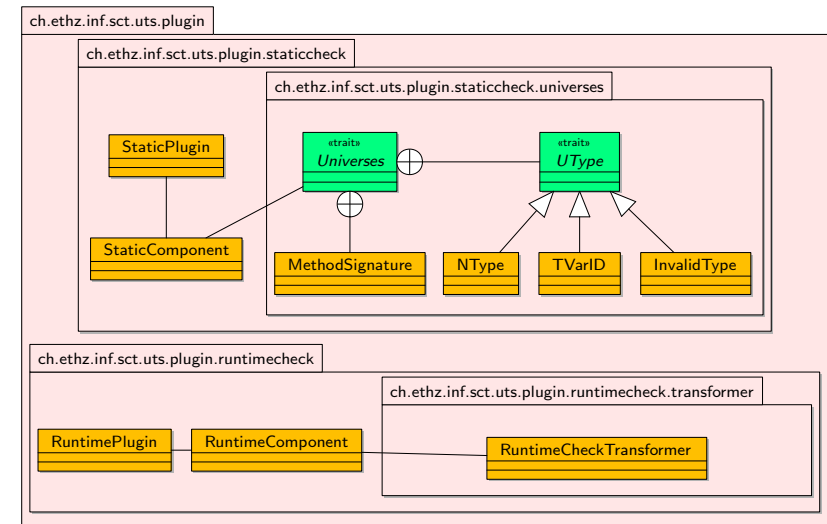
```
0 scalac -Xplugin-types \
  -Xplugin:<path-to-plugin.jar> \
2 -P:<plugin-name>:<plugin-option> <source files>
```

Inference of Universe Annotations

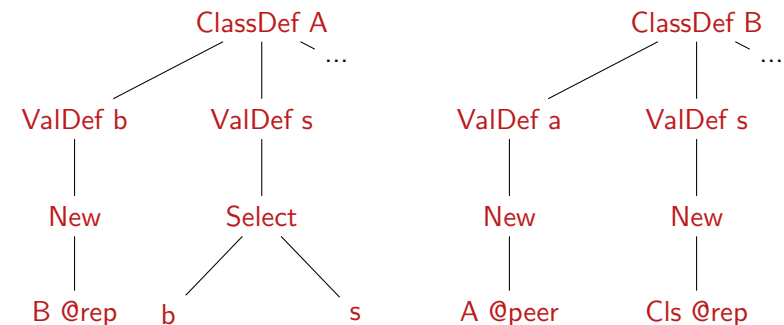
```
0 package ch.ethz.inf.sct.uts.examples
  import ch.ethz.inf.sct.uts.annotation._
2 class A {
    val b /* : B @rep */ = new (B @rep)
4   val s /* : Cls @any */ = b.s
    def t /* : Cls @peer */ = new (Cls @peer)
6   val u /* : Cls @rep */ = b.t
}
```

```
0 package ch.ethz.inf.sct.uts.examples
  import ch.ethz.inf.sct.uts.annotation._
2 class B {
    val a /* : A @peer */ = new (A @peer)
4   val s /* : Cls @rep */ = new (Cls @rep)
    val t /* : Cls @peer */ = a.t
6   val u /* : Cls @any */ = a.s
}
```

Universe Type System Plugins for Scala



Inference on the AST



Symbol		Typed	Locked	Code
A.b	⇒	false	false	ValDef b...
A.s	⇒	false	false	ValDef s...
B.a	⇒	false	false	ValDef a...
B.s	⇒	false	false	ValDef s...

Challenges and Problems

Conceptual

- ▶ Conversion of the type rules to actual code
- ▶ Finding good abstractions

Technical

- ▶ Documentation of the Scala compiler
- ▶ Initial handling of annotated types in the Scala compiler

Status and Next Steps

Next Steps



- ▶ Support for a broader set of Scala constructs:
 - ▶ Variance annotations for generics
 - ▶ Proper support for `if` and `match` control structures
- ▶ Case study with Scala Collection Library:
 - ▶ How far can I get with defaults and inference?
 - ▶ Which defaults require less hand-written annotations?
- ▶ Runtime support for generics
- ▶ Usage guide for the plugins, general report

Status and Next Steps

Status

- ▶ Superset of the language subset from [DDM07, Section 3.1] supported, including:
 - ▶ Constructors
 - ▶ Static objects
 - ▶ Inner classes
 - ▶ Basic variants of control structures
- ▶ Inference and propagation of Universe annotations works
- ▶ Inferred annotations are stored in the class files
- ▶ Basic runtime checks are added to the AST

References

-  W. Dietl, S. Drossopoulou, and P. Müller.
Generic Universe Types.
In E. Ernst, editor, *European Conference on Object-Oriented Programming (ECOOP)*, volume 4609 of *Lecture Notes in Computer Science*, pages 28–53. Springer-Verlag, 2007.
-  Daniel Schregenerberger.
Runtime Checks for the Universe Type System, 2004.
[Semester Thesis](#).

Basic Inference Algorithm

```
0  getMethodAndFieldSymbols(AST)

2  def process(AST) {
    if (access to untyped field/method symbol) {
4     inferType(symbol)
    }
6     propagateUniverseAnnotationsTo(AST)
    }
8
    def inferType(symbol) {
10     process(symbol.definition)
        copyUniverseAnnotations(symbol.definition,
12         symbol)
    }
```

Runtime Checks

- ▶ Basically a port of [Sch04] to Scala
- ▶ Store the owner of new objects during constructor execution
- ▶ Owner is stored in a hash table
- ▶ Additional checks are done on `isInstanceOf` and `asInstanceOf` calls