

From Scala to **Scala--**

© Miguel Garcia, STS, Hamburg University of Technology
<http://www.sts.tu-harburg.de/people/mi.garcia>

January 4, 2010

Contents

1	Motivation and Goals	1
2	Implementation of the current prototype	2
2.1	Block un-nesting	2
2.2	Recovering loops from <code>LabelDef-Apply</code> pairs	2
3	Work in progress	3

Abstract

The Scala compiler processes ASTs over a number of phases, where each phase recasts ASTs into a simpler language fragment. The last phase where the intermediate language is still a Scala subset is `CleanUp`, afterwards `ICode` is used. These DRAFT notes explore the design decisions required to rephrase (in terms of Scala textual syntax) the ASTs at the end of each non-`ICode` phase, in a manner that would allow using a Scala IDE on the resulting source files. For the most-reduced ASTs (at the end of the `CleanUp` phase) we dub the resulting syntax **Scala--**

1 Motivation and Goals

This exercise has three short term goals:

1. providing better insight into the transformations performed by each phase, by relying on the IDE facilities for code navigation and summarization (the user-friendliness of `-Xprint:<phase>` and `-Yshow-trees` notwithstanding :-)
2. from an instructional point of view, we also want to finally understand what all those AST shapes actually mean (e.g., the interplay between `LabelDef` and `Apply`).
3. provide yet another mechanism to check whether intermediate ASTs are well-formed (in a way more complete manner than with `-Ycheck`)

Goal (1) is not only relevant for compiler hackers, but for most Scala users. Otherwise, how can you know what AST rewritings a given compiler plugin performs? (e.g., to handle Continuations). Or will you inspect the resulting bytecode?

The implementation of the prototype generating `Scala--` is discussed starting in the next section. Figure 1 depicts a fragment of the same file (`Namers.scala`) in both full `Scala` and `Scala--`.

As a longer term goal, it is *thinkable* to develop a new backend for the `Scala` compiler, to generate Java 5 instead of bytecode. The compiler plugin realizing such backend could take as input either `ICode` ASTs (before `ICode`-level optimizations are performed); or `Scala--` ASTs. Admittedly, the ensuing Java compilation would most likely not be competitive with `scalac`, given that Java compilers have not been tuned to cope with the idioms present in `Scala--` (e.g., dead closure elimination). Or can they be incorporated into the repertoire of optimizations that Java compilers know about?

In any case, as a compiler-related project, the proposed backend is a worthwhile one. For example, does some phase produce irreducible loops? If so, can they be emulated in Java?

2 Implementation of the current prototype

Our prototype¹ (STST, for “Scala To Scala Translator”) relies on functionality originally developed to produce `Scala` documentation:

- from the `Scala X-Ray` compiler plugin, we adopt the workflow of traversing compilation units and generating a text file for each. The `X-Ray` output are HTML-based depictions of (full) `Scala` source code, with hyperlinks from uses to declarations, and additional mouse-over information.
- from the `TreePrinters` component of the `Scala` compiler, we customize the representation of tree nodes, so as to produce valid `Scala` code. For example, modifiers like `<synthetic>` are wrapped inside comment delimiters.

An introduction to `Scala X-Ray` appeared in Ch. 3 of the *Untangling Scala ASTs* notes². `X-Ray` is maintained at <http://github.com/harrah/browse>

2.1 Block un-nesting

Intermediate ASTs are usually more verbose than manually-written code, that’s why some transformations for readability prove useful (see *Constant Folding Redux*³). In addition, STST un-nests block of statements when they are always part of the same execution path (Listing 1).

2.2 Recovering loops from `LabelDef-Apply` pairs

The detection of while and do-while loops is done in `class MyTreePrinters(ofs : OutputStream) extends TreePrinters`, in particular at `recoverLoopStructure()` and `printLoop()`. The shapes of these trees was gleaned from the source snippets and comments shown in Listing 2. `LabelDef-Apply` nodes are introduced at least by the following :-) (1) by the parser, (2) during the `TailCalls` phase, (3) by the

¹`Scala-` at <http://www.sts.tu-harburg.de/people/mi.garcia/ScalaCompilerCorner>

²<http://www.sts.tu-harburg.de/people/mi.garcia/ScalaCompilerCorner/UntanglingScalaASTs1ofN.pdf>

³<http://www.sts.tu-harburg.de/people/mi.garcia/ScalaCompilerCorner/ReduxReport.pdf>

Listing 1: Un-nesting of statement blocks that stand in the way

```
case Block(stats, expr) if ((expr != EmptyTree) && stats.isEmpty) =>
  { someReductionApplied = true; transform(expr) }
case Block(stats1, b2 @ Block(stats2, expr2)) =>
  { someReductionApplied = true;
    val resTree =
      treeCopy.Block(tree,
        transformStats(stats1 ::: stats2 ::: List(expr2), currentOwner),
        EmptyTree)
    resTree
  }
case Block(b1 @ Block(stats1A, expr1A) :: stats1B, expr2) =>
  { someReductionApplied = true;
    val resTree =
      treeCopy.Block(tree,
        transformStats(stats1A ::: List(expr1A) ::: stats1B, currentOwner),
        transform(expr2))
    resTree
  }
```

translator of pattern matching. This list is non-exhaustive (additional places are reported by the IDE when searching for usages of the `LabelDef` constructor).

3 Work in progress

A lot! Not all loops are recovered (see those `goto` statements generated as comments?). Currently, STST can only be run after `CleanUp`.

On the plus side, IDE-debugging can be configured to step both plugin and compiler sources (as described for Eclipse⁴ and for IntelliJ⁵. More info on this at *The Scala Compiler Corner*, <http://www.sts.tu-harburg.de/people/mi.garcia/ScalaCompilerCorner>

⁴<http://comments.gmane.org/gmane.comp.lang.scala.tools/2113>

⁵<http://www.sts.tu-harburg.de/people/mi.garcia/ScalaCompilerCorner/img/HowToDebugCompilerPluginsInIntelliJ.png>

Listing 2: What is known about LabelDef and Apply pairs

```
// First, from Trees.scala, "A standard pattern match"
case LabelDef(name, params, rhs) =>
// used for tailcalls and like
// while/do are desugared to label defs as follows:
// while (cond) body ==> LabelDef($L, List(), if (cond) { body; L$() } else ())
// do body while (cond) ==> LabelDef($L, List(), body; if (cond) L$() else ())

// Second, from TreeBuilder.scala
/** Create tree representing a while loop */
def makeWhile(lname: Name, cond: Tree, body: Tree): Tree = {
  val continu = atPos(o2p(body.pos.endOrPoint)) { Apply(Ident(lname), Nil) }
  val rhs = If(cond, Block(List(body), continu), Literal(()))
  LabelDef(lname, Nil, rhs)
}
/** Create tree representing a do-while loop */
def makeDoWhile(lname: Name, body: Tree, cond: Tree): Tree = {
  val continu = Apply(Ident(lname), Nil)
  val rhs = Block(List(body), If(cond, continu, Literal(())))
  LabelDef(lname, Nil, rhs)
}
```

```

2 2  << package scala.tools.nsc {
3 3  << package typechecker {
4 4  << abstract trait Namers extends java.lang.Object with scalaObject { self: scala.tools
5 5  << /*accessor*/ def scalaTools$nsc$typecheckerNamers$setter_$classNameOfModuleCl
6 6  << /*accessor*/ def scalaTools$nsc$typecheckerNamers$setter_$classNameOfModule
7 7  << def newMember(context: scala.tools.nsc.typechecker.Context): scala.tools.nsc.typec
8 8  << /*stable*/ /*accessor*/ private[typechecker] def classNameOfModule(): scala.
9 9  << /*stable*/ /*accessor*/ private[typechecker] def classNameOfModule(): scala.C
10 10 << def resetMember(): Unit;
11 11 << def mkTypeCompleter(t: scala.tools.nsc.ast.Tree, c: Function1): scala.tools.nsc.t
12 12 << def underlying(member: scala.tools.nsc.symbtab.Symbol): scala.tools.nsc.symbtab.Sym
13 13 << def varNotice(sym: scala.tools.nsc.symbtab.Symbol): java.lang.String
14 14 << };
15 15 << abstract trait Namers$class extends { self: scala.tools.nsc.typechecker.Analyzer =
16 16 << def newMember($this: scala.tools.nsc.typechecker.Analyzer, context: scala.tools.nsc
17 17 << def resetMember($this: scala.tools.nsc.typechecker.Analyzer, context: scala.tools.nsc
18 18 << $this.caseClassOfModuleClass().clear();
19 19 << $this.classNameOfModule().clear();
20 20 << };
21 21 << def mkTypeCompleter($this: scala.tools.nsc.typechecker.Analyzer, tsi: scala.tools
22 22 << def underlying($this: scala.tools.nsc.typechecker.Analyzer, member: scala.tools.n
23 23 << if (member.isDeferred())
24 24 << {
25 25 <<   val getter: scala.tools.nsc.symbtab.Symbol = if (member.isSetter())
26 26 <<     member.getter(member.owner())
27 27 <<   else
28 28 <<     member;
29 29 <<   val result: scala.tools.nsc.symbtab.TermSymbol = getter.owner().newValue(get
30 30 <<   if (getter.setter(member.owner())) { $this.global().nscSymbol() }
31 31 <<   result.setFlag(4896)
32 32 <<   else
33 33 <<     scala.runtime.BoxedUnit.UNIT;
34 34 <<   result
35 35 << }
36 36 << else
37 37 <<   member.accessed()
38 38 <<   member;
39 39 << def varNotice($this: scala.tools.nsc.typechecker.Analyzer, sym: scala.tools.nsc.s
40 40 <<   ""012(note that variables need to be initialized to be defined)"
41 41 <<   else
42 42 <<     ""
43 43 << };
44 44 << def /*Name$class*/ $init($this: scala.tools.nsc.typechecker.Analyzer): Unit = {
45 45 <<   $this.scalaTools$nsc$typecheckerNamers$setter_$classNameOfModuleClass_=(new
46 46 <<   $this.scalaTools$nsc$typecheckerNamers$setter_$classNameOfModule_=(new s
47 47 <<   )
48 48 << }
49 49 << };
50 50 << class deskoLemizeMap extends scala.tools.nsc.symbtab.TypeMap with scalaObject {
51 51 <<   /*Name$accessor*/ private[this] val tpParams: List = ;
52 52 <<   def apply(tp: scala.tools.nsc.symbtab.Type): scala.tools.nsc.symbtab.Type = {
53 53 <<     var temp1: scala.tools.nsc.symbtab.Type = tp;
54 54 <<     if (temp1.$asInstanceOfOf[scala.tools.nsc.symbtab.TypeRef]())
55 55 <<     {
56 56 <<       var temp2: scala.tools.nsc.symbtab.TypeRef = temp1.$asInstanceOfOf[scala.tools
57 57 <<       var temp3: scala.tools.nsc.symbtab.Type = temp2.pre();
58 58 <<     }
59 59 <<     var temp4: scala.tools.nsc.symbtab.Symbol = temp2.sym();
60 60 <<     val temp5: List = temp2.args();
61 61 <<     {
62 62 <<       val args: List = temp5;

```

Figure 1: The Namers.scala file in full Scala (left) and Scala-- (right)