Dead Code Detection in Scala Compiler Source
SCALA COMPILER

- Written in Scala and Java
- Notoriously slow
- Fair amount of “dead code” in the scalac code base
- Highlight such code in the scalac source
Dead Code Detection Goals

1. Code which is unreachable and never executed.

2. Dead code which has no external impact

3. Dead code which may have external impact (ex. a division where the result is not used but would throw an exception on division by zero.)
DELIVERABLES

- A tool (source code and binary) that identifies the above three types of dead code in current version of Scala

- The result of applying this tool on the Scala compiler source
  - Give detailed line # and type of dead code

- A report that documents the details of your approach and your code
PROJECT PREVIEW PRESENTATION

- Mar 25
- 15-min presentation
- Your approach (more details than this lecture) and preliminary results
- Problems & Challenges
- Small demo
POSSIBLE APPROACHES

- Static analysis of source code (*deferred*)
  - Control flow analysis
  - Data flow analysis

- Dynamic analysis (testing)
  - Give many inputs
  - Trace the program to get data flow

- By building a scala compiler plugin based on Linter and the API of scala compiler

- By translating Scala code to Java and use Java static analysis tools.
Dynamic Analysis

- Debugger in Eclipse outputs execution trace as well as variable values at each step of the program
- Construct data flow diagram for each run
- After many runs:
  - Lines not executed at all $\rightarrow$ type 1 dead code
  - Lines executed but didn’t contribute to the final output $\rightarrow$ type 2 dead code
  - Lines executed and didn’t contribute to the final output but contains side effects $\rightarrow$ type 3 dead code
**COMPILER PLUGINS**

- Scala’s compiler can do part of dead code detection too by adding options “-Ywarn-dead-code”.

- **Linter** is a Scala static analysis compiler plugin which adds compile-time checks for various possible bugs, inefficiencies, and style problems.
What can Scala compiler do?

- Data flow analysis

- Mark and Sweep algorithm to detect and remove part of dead code.
  - Unreachable code
  - Useless dead code

- But this is done at the three-address code (IR) level.

- Need a way to map the instructions back into source code statements/expressions.
WHAT CAN SCALA COMPILER DO?

- In this way, it should detect some unreachable code and dead code.

- For example, it can detect dead code after return statement.

- But unfortunately, it doesn’t even warn the unused assignment as dead code.
UNREACHABLE CODE

def somethingAfterReturn(): Int = {
    println("That is ok..");
    return 1;
    println("What is going on here?");
    3
}

scala> def somethingAfterReturn():Int = {
    | println("That is ok..");
    | return 1;
    | println("What is going on here?");
    | 3
    | }
<console>:9: warning: dead code following this construct
    | return 1;
    | ^
somethingAfterReturn: ()Int
scala>
WHAT CAN LINTER DO?

- It can detect some unreachable code and some useless executions as special cases, which is more powerful than the optimization in scalac.
def somethingUnreachable1(): Int = {
    println("That is ok..");
    val x = 1
    if (x > 0){
        println("That is ok..");
        if (x < -1){
            println("This is dead code!");
        }
    }
    return 2;
}
Linter

INFO: compiling 1 Scala source to 1 Scala class target Scala core classes...

[warn] D:\scala\hello\hw.scala:11: a pure expression does nothing in statement position; you may be omitting necessary parentheses
  [warn]     return 2;
  [warn]     ^
[warn] D:\scala\hello\hw.scala:11: enclosing method main has result type Unit: return value discarded
  [warn]     return 2;
  [warn]     ^
[warn] D:\scala\hello\hw.scala:5: This condition will always hold.
  [warn] if (x > 0){
  [warn]     ^
[warn] D:\scala\hello\hw.scala:7: This condition will never hold.
  [warn] if (x < -1){
  [warn]     ^
[warn] four warnings found
[info] Running HelloWorld
That is ok..
That is ok..
[success] Total time: 3 s, completed 2014-12-15 8:09:26
def somethingUnreachable2(): Int = {
    var x = 1
    var y = 0
    if (y== 0){
        println("This is y==0!")
        if (y == -1)
            println("This is y!=0!")
    }
    if (x!=1)
        y = x/x
}

Linter

```scala
> run
[info] Compiling 1 Scala source to D:\scala\hello\target\scala-2.10\classes...
[warn] D:\scala\hello\hw.scala:5: These two nested ifs can be merged into one.
[warn] if (y==0)
[warn] ^
[warn] D:\scala\hello\hw.scala:5: This condition will always hold.
[warn] if (y==0)
[warn] ^
[warn] D:\scala\hello\hw.scala:8: This condition will never hold.
[warn] if (x!=1)
[warn] ^
[warn] three warnings found
[info] Running HelloWorld
[succeed] Total time: 8 s, completed 2014-12-16 0:31:27
```
WHAT CAN LINTER DO?

- It will test each known Boolean expression and warn you which part is unreachable.

- It can do more, such as unused method parameters, identical branch and etc.
**Linter/Scala Build Issues**

- Linter is built under SBT
- But scalac is currently built under ANT
- Need a way to transform the current scala build.xml to SBT so that it can be compiled under SBT
POSSIBLE STEPS IN MODIFIED LINTER

1. Compile into bytecode; instructions annotated with source code locations.

2. For type 1 dead code, we can use execution flow (control flow) analysis combined with linter compiler plugins to find unreachable code.

3. For type 2 dead code, we use the mark and sweep algorithm, the same as used in scala compiler to accurately find dead code.
POSSIBLE STEPS IN MODIFIED LINTER

4. For type 3 dead code, instructions identified by step 3 which have side effects (IO or exceptions) are marked as type 3 dead code.

5. For some special case, deal with them individually.

6. Return source code locations of all identified instructions.
TRANSLATION INTO JAVA

1. Compile scala program by “scalac” into a java .class file (bytecode), with annotation of source locations.
2. Use a decompiler (Javap or JDEclipse) to decompile .class file back into Java.
3. Establish mapping between scala source and Java source.
4. Use a dead code detector (UCDetector) to detect dead code (classified into 1 of the 3 types)
5. Map back to scala source code.
WHAT UCDetector can do

- For the first type, it can be detected by compilers.
WHAT UCDetector CAN DO

- Any thing in the “if” block is exempted from dead code detection.

```
package tmp;

public class Main {
    public int tmp() {
        int a=1,b=0,c=0;
        if (a>0) {
            c=0;
            if (a<0) b=1;
        }
        return b+c;
    }
}
```
What UCDetector can do

- For the second type, it can also be detected.
WHAT UCDetector can do

- For the third type, it can’t be detected at all.

```java
package tmp;
import java.util.Scanner;
public class Main {
    public int tmp() {
        Scanner cin=new Scanner(System.in);
        int a,b,c,d;
        a=cin.nextInt();
        b=cin.nextInt();
        c=a/b;
        d=a+b;
        return d;
    }
}
```
DECOMPILING IN JAVA

- There are many decompilers for Java such as javap.

- After decompiling, we get a list of declarations in Java formats and some functions remain in byte codes.

- We can change the byte codes into source codes by referencing the language specification.

- But it may be much harder than the first approach.
EXAMPLE OF BYTE CODES

```java
public void init();

Code:
  0: aload_0
  1: sipush 500
  4: bipush 100
  6: invokevirtual #2    // Method resize:(II)V
  9: aload_0
 10: aload_0
 11: ldc    #3    // String LAST_UPDATED
 13: invokevirtual #4    // Method getParameter:(Ljava/lang/String;)Ljava/lang/String;
 16: putfield   #5    // Field date:Ljava/lang/String;
 19: aload_0
 20: aload_0
 21: ldc    #6    // String EMAIL
 23: invokevirtual #4    // Method getParameter : (Ljava/lang/String;)Ljava/lang/String;
 26: putfield   #7    // Field email:Ljava/lang/String;
 29: return
```
Example

- Source code in scala:

```scala
object Main {
    def main(args: Array[String]) = {
        println("Hello, " + args(0))
    }
}
```
**Example**

- Javap output in Main$.class:

```java
public final class Main$ extends java.lang.Object
    implements scala.ScalaObject{
    public static final Main$ MODULE$;
    public static {};
    public void main(java.lang.String[]);
}
```
**Example**

- Javap output in Main.class:
  ```java
  public final class Main extends java.lang.Object{
    public static final void main(java.lang.String[]);
  }
  ```
- As we can see it’s the same as we can get when compiling a java file.
CHALLENGE 1

- Decompilation tools for Java can be incomplete and unreliable.
CHALLENGE 2

- After compiling and decompiling, we can get source codes in java but not in scala.
- Possible solution: mapping source codes from java to scala or from byte codes to scala with the help of compiler by changing the compiler.
- Also, the decompiling is not explicit.