Natural Language Programming Analysis in Scala

Éric Zbinden
eric.zbinden@epfl.ch

Supervisors:
Philippe Sutter(philippe.sutter@epfl.ch)
Philipp Haller(philipp.haller@epfl.ch)
Prof. Viktor Kuncak(viktor.kuncak@epfl.ch)

EPFL
Laboratory for Automated Reasoning and Analysis (LARA)
http://lara.epfl.ch/
Programming Methods Laboratory (LAMP)
http://lamp.epfl.ch/

January 6, 2012

Contents

1 Introduction

...
For Java this was done by

2 Project Overview

The main idea of this project is to apply an analysis similar to Høst and Østvold [4] but to Scala. Scala contains possibilities that JAVA don't contains; functional programming, local import. So I tried to focus on theses particularities as much as possible.

3 Implementation

The first thing is to retrieve information about a code. The best way is to create a plug-in for the Scala compiler.

3.1 Compiler plug-in

My plug-in, named Scala-names, is inserted into the compiler phases right after phase 7: *refchecks*. It use the abstract syntax tree produced in previous phases to extract all objects named by the programmer. It could be variables names, objects names, methods names, parameters names or types names.

By lack of time, the analysis is run only on method names. But the plug-in could easily be improved to also analyze other objects. The plug-in then check for every method definition found if that method have a given list of features. See 3.2 for more details on features. Then the plug-in output for every method a list of "1" or "0" depending on this method satisfy or not a feature and the method name with source and position.

3.2 Features

A feature is a boolean property that a method may satisfy or not.

1. Return type is a collection :
2. Return type is a subclass of Object :
3. Return type is Unit :

4. Return type is boolean :

5. Return type is an Integer :

6. Return type is a String :

7. Method take no parameter :

8. Method is declared without parenthesis :

9. Method body contains IF statement :
   The method body contain an IF branch. Note that the Scala compiler translate WHILE block and DO-WHILE block with a IF branch. These two case should not match this feature as the programmer did not write any IF branch. IF guard statement in pattern matching are not taken in account as IF branch. However an IF branch inside a pattern matching right hand side will match.
   IF branch matching inside pattern matching right hand side:

```
ls match {
  case Nil => false
  case x :: xs => if(x==0) true else false
}
```

   IF guard in pattern matching

```
ls match {
  case Nil => false
  case x :: xs if(x==0) => true
  case x :: xs => false
}
```

10. Method body contains WHILE statement :
    The method body contains a WHILE statement. DO-WHILE statement are also considered as WHILE statement.

11. Method body contains TRY-CATCH statement :
    The method body contains a TRY-CATCH block. If a method body contains only the TRY block, it will also match.

12. Method body contains pattern matching :
    The method body contains pattern matching. If match if the method body contains a MATCH statement.

13. Method body contains explicit THROW statement :
    The method body contains a THROW statement. It match only if it's explicitly declared. A NumberFormatException raised by a wrong string applied to *.toInt* will not match.

14. Method is currified :

15. Method is self-recursive :
    This method call it-self in it's body. The feature don't match for other method even with surcharged identifiers.

16. Method name is a verb :

17. Method name is a noun :

18. Method name is a camel phrase :

19. Method name contains an acronym :

20. Method name match an abstract phrase construction :

21. Method name contains "is" pattern :

22. Method name contains "get" pattern :
    Straight forward, it's implemented the same way as feature 21.

23. Method name contains "set" pattern :
    Same as features 21 and 22.

24. Method name contains "contains" pattern :
    Same as features 21, 22 and 23.

25. Method name is a valid JAVA name :
    A valid JAVA method name is a series of JAVA letters or JAVA digit that begin with a JAVA letter [2] and that is not a JAVA keyword [3].

26. Method name is an operator :
    An operator is defined as a following of characters that are neither a letter neither a digit.

27. Method return type is completely contained into the method name :

28. Method return type is partially contained into the method name :
    This feature is a variant of feature 28. For composed type name, as programmer are often lazy, they often write it partially. One would by example

write only "Tree" instead of "AbstractSyntaxTree". This feature will match in such case as previous feature will not. It will also match if the type is completely contained, like 28.

29. Method is right associative :
30. Method is declared into another method :
31. Method body contains inner method definition :
32. Method is overriding another method :
33. Method is abstract :
34. Method is public :
35. Method is static :
36. Method name finish with "s" :
37. Method name finish with "ss" :

3.3 Clustering

With the output of the Scala-names plug-in, I filled the k-means algorithm [1] to find out something interesting. The implementation of the algorithm follow [1] description with a random partition.

4 Experimental Results

4.1 Features

4.2 Correlation

4.3 Others

5 Conclusion

6 Future and Related Work

Add new features, look at object, trait and class. Parameters and variables.

References

[1] k-means clustering. consulted January 5, 2012, http://en.wikipedia.org/wiki/K-means_clustering.

[2] Java identifiers. In Microsystems [5]. http://java.sun.com/docs/books/jls/third_edition/html/lexical.html#3.8.

[3] Java keywords. In Microsystems [5]. http://java.sun.com/docs/books/jls/third_edition/html/lexical.html#3.9.

[4] Einar W. Høst and Bjarte M. Østvold. Debuging method names. In *Proceedings of 2009 ECOOP - Object-Oriented Programming*, Genova, Italy, July 6–10 2009.

[5] Sun Microsystems, editor. Santa Clara, California, USA, 2005.