scalafmt: opinionated code formatter for Scala

Ólafur Páll Geirsson

School of Computer and Communication Sciences

Master's Thesis

June 2015

Responsible
Prof. Martin Odersky
EPFL / LAMP

Supervisor
Eugene Burmako
EPFL / LAMP

Abstract

Automatic code formatters bring many benefits to software development, yet they can be tricky to implement. This thesis addresses the problem of developing a code formatter for the Scala programming language that captures many popular coding styles. Our work has been limited to formatting Scala code. Still, we have developed algorithms and tools, which we believe can be of interest to developers of code formatters for other programming languages.

Contents

1 Introduction

The main motivation of this study is to bring scalafmt, a new Scala code formatter, to the Scala community. The goal is to capture many popular coding styles so that a wide part of the Scala community can enjoy the benefits that come with automatic code formatting.

Without code formatters, software developers are responsible for manipulating all syntactic trivia in their programs. What is syntactic trivia? Consider the Scala code snippets in listings 1 and 2.

Listing 1: Unformatted code

```
1  // Column 35                          |
2  object ScalafmtExample {
3    function(arg1, arg2(arg3(
4      "String literal", arg4, arg5),
5      arg6 + arg7))
6  }
7
8
9
10
```

Listing 2: Formatted code

```
1  // Column 35                          |
2  object ScalafmtExample {
3    function(
4        arg1,
5        arg2(arg3("String literal",
6                  arg4,
7                  arg5),
8             arg6 + arg7))
9  }
10
```

Both snippets represent the same program. The only difference lies in their syntactic trivia, that is where spaces and line breaks are used. Although the whitespace does not alter the execution of the program, listing 2 is arguably easier to read, understand and maintain for the software developer. The promise of code formatters is to automatically convert any program that may contain style issues, such as in listing 1, into a readable and consistent looking program, such as in listing 2. Automatic code formatting offers several benefits.

Code formatting enables large-scale refactoring. Google used ClangFormat[6], a code formatter, to migrate legacy C++ code to the modern C++11 standard[13]. ClangFormat was used to ensure that the refactored code adhered to Google's extensive C++ coding style[4]. Similar migrations can be expected in the near future for the Scala community once new dialects, such as Dotty[9], gain popularity.

Code formatting is valuable in collaborative coding environments. The Scala.js project[11] has over 40 contributors and the Scala.js coding style[12] contains over 2.600 words. Each contributor to the Scala.js project is expected to follow the coding style. Each contributed patch is manually verified against the coding style by the project maintainers. This adds a burden on both contributors and maintainers. Several maintainers of popular Scala libraries have expressed this sentiment. ENSIME[2] is a popular Scala interaction mode for text editor. Sam Halliday, a maintainer of ENSIME, says "I don't have time to talk about formatting in code reviews. I want the machine to do it so I can focus on the design."[5]. Akka[1] is another Scala library to build concurrent and distributed applications. Viktor Klang, a maintainer

of Akka, suggests a better alternative "Code style should not be enforced by review, but by automate rewriting. Evolve the style using PRs against the rewriting config.".[7]. With code formatters, software develoeprs can direct their full focus on writing correct, maintainable and fast code.

1.1 Whitespace style issues

Mechanical style issues are source code issues that can be fixed automatically. For example, consider listing 3.

Listing 3: Style issues

```
1 if  ( condition(  predicate)) {
2     println("Goodbye" );
3   System.exit(1) }
```

This code snippet leaves a lot left to be desired. Mechanical issues include the redundant spaces around parentheses, unnecessary semicolon after the `println` and the inconsistent indentation in the body of the if statement. Non-mechanical issues may include the fact the program prints an error message to the standard output and exits the process, instead of throwing an exception.

This thesis only addresses the whitespace style issues. That is, to automatically fix the spaces and newline characters between the non-whitespace tokens in the original source code. We will leave it to the software developer to decide whether the unnecessary semicolon should remain in the source file or if the program should throw an exception.

1.2 Contributions

The main contribution presented in this thesis are the following:

- scalafmt, a code formatter for the Scala programming language. At the time of this writing, scalafmt has been available for 3 months, it has been installed over 5.000 times and is already in use by several open source Scala libraries. For details on how to install and use scalafmt, refer to the scalafmt online documentation[3].
- algorithms and data structures to implement line wrapping under a maximum column-width limit. This work is presented in section 3.
- tools to develop and test code formatters. This work is presented in section 4.

The scalafmt formatter itself may only be of direct interest to the Scala community. However, much work in this thesis is not specifically tied to Scala and we hope can inspire the design of code formatters for other programming languages.

2 Background

This chapter explains the necessary background to understand Scala and code formatting. More specifically, we go into details on Scala's rich syntax and popular Scala idioms which introduce unique challenges to the design of a code formatter for Scala. We follow up with a brief history on code formatters for both Scala as well as other programming languages. We will see that although code formatters have a long history, the last 5 years have brought a lot of research in optimization based formatters, which scalafmt's design takes inspiration from.

2.1 Scala the programming language

Scala[8] is a general purpose programming language that was first released in 2004. Scala combines features from object-oriented and functional programming paradigms, allowing for concise and high-level programming. Most commonly, Scala programs compile to bytecode and run on the JVM. With the releases of Scala.js[12], JavaScript has recently become a popular target platform for Scala developers. Even more recently, the announcement of Scala Native[10] shows that LLVM and may become yet another viable platform for Scala developers. The Scala Center estimates that more than half a million developers are using Scala[**scala-center**].

2.1.1 Pattern matching

2.1.2 Higher order functions

2.1.3 For comprehensions

2.1.4 Dialects

2.1.5 Metaprogramming with scala.meta

- Tree nodes have parent links.
- Token classes are types.

2.2 Code formatting

2.2.1 gofmt

2.2.2 rustfmt

2.2.3 dartfmt

2.2.4 clang-format

2.2.5 scalariform

There already exists a widely used code formatter for Scala called Scalariform[**scalariform**]. However, Scalariform lacks the ability to enforce a column width limit, which we consider necessary to capture many popular Scala coding styles.

3 Algorithms

3.1 Data structures

3.1.1 FormatToken

3.1.2 Split

3.1.3 Policy

3.1.4 OptimalToken

3.1.5 State

3.1.6 Indent

3.2 BestFirstSearch

3.2.1 Router

- Router
- Search.

3.3 Optimizations

3.3.1 dequeueOnNewStatements

3.3.2 recurseOnBlocks

3.3.3 escapeInPathologicalCases

3.3.4 escapeInPathologicalCases

3.3.5 pruneSlowStates

3.3.6 FormatWriter

- vertical alignment
- comment formatting
- stripMargin alignment

4 Tooling

4.1 Heatmaps

4.2 Configuration

4.2.1 maxColumn

4.2.2 binPacking

4.2.3 vertical alignment

4.3 Unit tests

4.4 Property based tests

4.4.1 AST Integrity

4.4.2 Idempotency

4.5 Regressions tests

5 Evaluation

5.1 Micro benchmarks

5.2 Adoption

6 Discussion

6.1 Future work

6.2 Conclusion

References

[1] *Akka*. URL: http://akka.io/ (visited on 05/29/2016).

[2] *ENSIME*. URL: http://ensime.github.io/ (visited on 05/29/2016).

[3] Olafur Pall Geirsson. *Scalafmt - code formatter for Scala*. URL: http://scalafmt.org (visited on 05/29/2016).

[4] *Google C++ Style Guide*. URL: https://google.github.io/styleguide/cppguide.html (visited on 05/28/2016).

[5] Sam Halliday. *I don't have time to talk about formatting in code reviews. I want the machine to do it so I can focus on the design.* microblog. May 2016. URL: https://twitter.com/fommil/status/727879141673078785 (visited on 05/29/2016).

[6] Daniel Jasper. *clang-format*. URL: http://llvm.org/devmtg/2013-04/jasper-slides.pdf (visited on 04/20/2016).

[7] Viktor Klang. *Code style should not be enforced by review, but by automate rewriting. Evolve the style using PRs against the rewriting config.* microblog. Feb. 2016. URL: https://twitter.com/viktorklang/status/696377925260677120 (visited on 05/29/2016).

[8] Martin Odersky et al. *The Scala language specification.* 2004. URL: http://www-dev.scala-lang.org/old/sites/default/files/linuxsoft_archives/docu/files/ScalaReference.pdf (visited on 05/31/2015).

[9] Tiark Rompf and Nada Amin. "From F to DOT: Type Soundness Proofs with Definitional Interpreters". In: *arXiv:1510.05216 [cs]* (Oct. 2015). arXiv: 1510.05216. URL: http://arxiv.org/abs/1510.05216 (visited on 05/28/2016).

[10] *scala-native/scala-native.* URL: https://github.com/scala-native/scala-native (visited on 05/29/2016).

[11] *Scala.js.* URL: http://www.scala-js.org/ (visited on 05/29/2016).

[12] *Scala.js coding style.* URL: https://github.com/scala-js/scala-js/blob/master/CODINGSTYLE.md (visited on 05/28/2016).

[13] Hyrum Wright et al. "Large-Scale Automated Refactoring Using ClangMR". In: (2013). URL: https://research.google.com/pubs/pub41342.html (visited on 04/21/2016).