scalafmt: yet another approach to code formatting

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 their are tricky to get right. This thesis addresses the problem of developing a code formatter for the Scala programming language that captures the language's most popular idioms and coding styles. Although our work has been limited to formatting Scala code, we have developed data structures, algorithms and tools that we believe may facilitate the creation of code formatters for a variety of other programming languages.

Contents

Throughout the paper we assume familiarity with the basics of the Scala Programming Language [1].

1 Introduction

A good code formatter When done right, code formatters relieve the developer's attention from manipulating syntactic trivia while helping enforce a consistent coding style in codebases.

Code formatting brings many benefits to software development.

Code formatting aids readability of code.

Code formatting enforces a consistent coding style.

Code formatting allows developers to put their focus on what matters.

Code formatting enables automated refactoring tools. See [2]

Code formatting lowers the barrier to entry for novice programmers. Take away the decision on where to insert spaces and newlines.

Code formatters are all or nothing. Due to their nature, either you use it for everything or for nothing.

1.1 Problem statement

- Opinionated, can produce nice looking code even from computer generated input.
- Fast, must not run for longer than X per LOC.
- Support popular coding styles of which most important is line length limit.

1.2 Contributions

The main contributions presented in this thesis are the following:

- Data structures, algorithms and tools for implementing advandes code formatters for syntatically rich languages.
- scalafmt, a case study where the framework is used to format Scala programs.

Non-goals:

- Language independent pretty printing. Hasn't gained popularity.

2 Related work

2.1 Combinator based

1. Houghes 1995
2. Wadler 1999

2.2 Optimization-oriented

1. clang-format Dijkstra's 2010
2. dartfmt Best-first search 2014
3. rfmt 2015

1. Optimal line breaking
2. Oppen

3 Background

3.1 Code formatting

3.2 Scala the programming language

3.2.1 Higher order functions

3.2.2 Pattern matching

3.2.3 Metaprogramming with scala.meta

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

4 Framework

4.1 Data structures

4.1.1 FormatToken

4.1.2 Split combinator

4.1.3 Policy combinator

4.1.4 OptimalToken combinator

4.1.5 State

4.1.6 Indent

4.2 Algorithms

Pipeline of three stages: pre-processing, line wrapping and post-processing.

4.2.1 Pre-processing

- AST vs. token stream

rustfmt considers that all eventually converge to a hybrid of the two. Indeed, clang-format has an elaborate parser.

4.2.2 Layout

- Router
- Search.

4.2.3 Post-processing

- vertical alignment
- comment formatting
- stripMargin alignment

4.3 Tooling

4.3.1 Heatmaps

4.3.2 Unit tests

4.3.3 Property based testing

1. AST integrity
2. Idempotent

5 scalafmt

5.1 Custom optimizations

1. dequeueOnNewStatements
2. recurseOnBlocks
3. escapeInPathologicalCases
4. pruneSlowStates

6 Evaluation

6.1 Performance

6.2 Output

- gofmt
- rustfmt
- dartfmt
- clang-format
- scalariform

7 Discussion

7.1 Future work

7.2 Conclusion

References

[1] 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).

[2] 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).