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 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. Our work has been limited to formatting Scala code. Still, we have developed data structures, algorithms and tools that we believe can applied to develop code formatters for a variety of other programming languages.

Contents

1 Introduction

Code formatters are used by software developers to automatically fix common formatting errors in their source code. Code formatters should not alter the behavior of the source code, only its aesthetics. Take for example this piece of code:

Listing 1: Unformatted code

```
object A {function(arg1, arg2(arg3(arg4, arg5, "arg6"), arg7 + arg8), arg9.select(1,
      2, 3, 4, 5, 6))}
```

Listing 2: Formatted code

```
object A {
  function(arg1,
           arg2(arg3(arg4, arg5, "arg6"), arg7 + arg8),
           arg9.select(1, 2, 3, 4, 5, 6))
}
```

Software developers use code formatters to automatically reformat their source code. A code formatter will typically fix common whitespace errors such as indentation, spaces and newline breaks. A key feature of code formatters is that they do not change the behavior of the code, only its aesthetics.

Software developers use code formatters to automatically fix white A code formatter should not change the behaviour of code. Typically, a code formatter only modifies whitespace tokens, or syntactic trivia. Syntactic trivia has nothing to do with the behavior of code. Code formatters promise to relieve software developers from manually manipulating syntactic trivia. Instead, developers can focus on writing correct, performant and maintainable code.

Code formatters brings many other benefits to software development. They can help enforce a consistent coding style in a codebase that's touched by multiple developers. A consistent code style aids readability of code. Code formatting also enables automated refactoring tools, see [**wright_large-scale_2013**]. Code formatting also lowers the barrier to entry for novice programmers, who are not aware of the coding style conventions.

Code formatters have existed for a long time. In more recent years we've seen many new opimization based code formatters, such as clang-format and dartfmt. Optimization based code formatters define a fit function for a given layout. A good layout has low cost while a bad layout has a high cost. The formatter runs an algorithm to choose the layout with the lowest code. This idea is not new, for example TeX uses a fit function to optimally break lines in text documents.

Much work has been put into developing language agnostic code formatters or pretty printers. Researchers of such work correctly highlights that developing a custom code formatter for each language requires a lot of effort. It may be possible that we find a perfect way to develop language independent

code formatters, see [**mps_article**]. Still, today no such framework exists and people who write Scala code each day are still choosing to manually format their code. The goal of this thesis is not to address language agnostic code formatting.

1.1 Problem statement

This thesis addresses the problem of developing a code formatter, `scalafmt`, for the Scala programming language. This formatter should:

- capture Scala's most common idioms and popular coding styles, most importanly a line length limit.
- be opinionated, it should be able to produce nice looking code even from computer generated input.
- be fast, <200ms for 99 percentile of source files, <2s for rest.

1.2 Contributions

The main contributions presented in this thesis are the following:

- We have developed a framework of data structures, algorithms and tooling that help developing code formatters. This work is presented in section X.
- We have applied this framework to develop `scalafmt`, a code formatter for the Scala programming language.

2 Background

2.1 Scala the programming language

Throughout the paper we assume familiarity with the basics of the Scala Programming Language [**odersky_scala_2004**].

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

3 scalafmt

3.1 Data structures

3.1.1 FormatToken

3.1.2 Split combinator

3.1.3 Policy combinator

3.1.4 OptimalToken combinator

3.1.5 State

3.1.6 Indent

3.2 Algorithms

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

3.2.1 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.

3.2.2 BestFirstSearch

- Router
- Search.

3.2.3 FormatWriter

- vertical alignment
- comment formatting
- stripMargin alignment

3.3 Heatmaps

3.4 Configuration

3.4.1 maxColumn

3.4.2 binPacking

3.4.3 vertical alignment

3.5 Optimizations

3.5.1 dequeueOnNewStatements

3.5.2 recurseOnBlocks

3.5.3 escapeInPathologicalCases

3.5.4 escapeInPathologicalCases

3.5.5 pruneSlowStates

3.6 Unit tests

3.7 Property based tests

3.7.1 AST Integrity

3.7.2 Idempotency

3.8 Regressions tests

4 Evaluation

4.1 Micro benchmarks

4.2 Adoption

5 Related work

5.1 Combinator based

1. Houghes 1995
2. Wadler 1999

5.2 Optimization-oriented

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

1. Optimal line breaking
2. Oppen

6 Discussion

6.1 Future work

6.2 Conclusion