Chapter 1
Related Work

Library-specific linters are rare because they require a non-trivial amount of work to implement. Even more niche is linters for embedded DSLs. This thesis can also be seen as a rough framework for implementing a linter for an eDSL (esp. Parser datatype to represent DSL model). afaik nothing this complex has been done in scalafix

Related to scalafix, so it could be possible to do very similar things as I've done: Rust (I can see this as possible, ask Boogle people for thoughts?) – Clippy lint passes C# – Roslyn

Library-specific linters: xunit.analyzers – seems to be relatively simple one-off rules though, not a DSL

Things that could help: Squid quasiquotes: type-safe and hygienic quasiquotes – but only for scala 2.11/12 * squid quasiquotes have rudimentary function inlining features https://infoscience.epfl.ch/record/231700 Scala 3 macros: I wonder if this could help?

Related concepts to the Function stuff: Metaprogramming and multi-staged programming? a dual: instead of for optimisation/code-generation purposes, we do it just for stringifying Actually I feel like I've borrowed a lot of stuff from metaprogramming optimisation techniques – Haskell Parsley itself

The leftrec transformation is not new but its implementation as a linter rule is novel

Chapter 2
Conclusion

2.1 Review

The left-recursion factoring transformation could be implemented in a staged version of `parsley` to eliminate left-recursion at compile-time with no run-time overhead. However, from a design perspective this is potentially controversial as it obfuscates the PEG semantics of the parser combinator library, allowing CFG-like parsers to be written directly. One could also argue (pedagogical perspective? not really) that the higher-level abstraction of the `chain` combinators is a more desirable form to write parsers for left-associative operations, rather than a lower-level left-recursive grammar rule.

2.2 Future Work

Separate into practical (improvements to parsley-garnish) and theoretical (cool research things)?

2.2.1 Expression AST

Eta reduction – this is more complicated than in Haskell since Scala has special syntax Proper partial evaluation, not just normalisation (if we reduce to fully closed terms 1+1 can we get it to evaluate to 2? – except currently this would be a Translucent term)

2.2.2 Resugaring

https://dl.acm.org/doi/10.1145/2858949.2784755