A formally verified equational theory of modern regular expressions

Supervised by Aurèle Barrière, Clément Pit-Claudel and Gabriel Radanne

Internship information

Location: Lyon, France.

Advisors: Aurèle Barrière, Clément Pit-Claudel, and Gabriel Radanne.

Date: Spring and summer 2026.

Contact: `aurele.barriere@epfl.ch`, `clement.pit-claudel@epfl.ch`, and `gabriel.radanne@inria.fr`

Summary

Modern *regexes* (the descendants of regular expressions) and their semantics are surprisingly complex. For instance, it could be tempting to believe that, for any regex `r`, the regex `r?` is equivalent to the regex `r|`ε: both regexes try to match `r` either 1 or 0 time. Several papers have even assumed so in the past [8, 3]. But in fact, this is incorrect for some JavaScript regexes [4]. Similarly, it looks like `r{n1,m1}r{n2,m2}` should be equivalent to `r{n1+n2,m1+m2}`: in fact, the `regexp-tree` manipulation library performs this optimization. But again, this is incorrect!

The goal of this internship is to study JavaScript regex equivalence, and develop a formally verified regex optimizer.

Background

Modern Regexes

Modern regexes have changed a lot compared to their ancestors, the well-known traditional regular expressions. For instance, JavaScript regexes include new complex features (backreferences, capture groups, lookarounds, ...). With these features, the regex matching problem becomes harder (it is in fact NP-hard [5]), known properties no longer hold true (for instance, disjunction is not commutative), and regex equivalence is no longer decidable [7].

This new complexity has not been without consequences: leading regex engines and regex manipulation libraries have bugs [1, 2].

Regex Transformations

In this project, we focus on transformations that rewrite a regex into an equivalent one. Such regex transformations have multiple real-world uses. First, regex optimizing libraries (like `regexp-tree` [10] for JavaScript regexes) rewrite regexes into equivalent, optimized regexes that are expected to be faster to match. Second, regex transformations can also be used internally by regex engines to handle some features or optimize some common cases (for instance, the V8 JavaScript engine desugars some quantifiers like `+` [1], replacing `r+` with `rr*`).

To avoid bugs in engines and in optimizers, it is important to ensure that these transformations are correct. This internship consists in developing a new theory of modern regular expression equivalence, then using this theory to develop a formally verified JavaScript regex optimizer.

Previous Work

This internship fits into a bigger project: we already have developed preliminary tools and techniques that can be used to formally verify regex equivalence. Specifically, we have previously:

- Translated into Rocq the regex chapter of the official JavaScript standard [6]. This is the Warblre [4] mechanization.
- Built a formally verified method to prove JavaScript regex equivalence [2] in Rocq.
- Used this method to verify a couple of regex rewrites used by the `regexp-tree` optimizer.

[1] https://github.com/v8/v8/blob/8ca0c3133f4785519c19b5c0f6287fa000c1cf4a/src/regexp/experimental/experimental-compiler.cc#L1099

Internship

Goals

- Develop a formally verified equational theory for modern regex equivalence. Our notion of *contextual equivalence* [2] is a good starting point, but needs to be adapted for transformations that can duplicate capture groups. One also needs to define helpful tactics and better reasoning principles for regex equivalence proofs.
- Use this theory to verify regex rewrites taken from existing optimizers like `regexp-tree` or the literature [9].
- Build a formally verified regex optimizer that repeatedly applies these regex rewrites.
- Define a new intermediate representation (IR) for regexes, allowing more regex rewrites. For instance, some engines use some transformations only when the regex has been compiled to an NFA. We believe that an alternative approach could consist in translating the regex into a new IR, performing verified rewrites in that new IR, then finally translating this IR into an NFA. Rewriting transformations could be easier to select and apply on the AST of an IR than at the NFA level.

Interesting optional research directions also include:

- Proving the completeness of the approach. This means proving that if two subregexes are not equivalent according to our criterion, then there exist a regex context and an input string for which they return different results.
- Use your formally verified optimizer as the first step of our formally verified regex engine [2]. This means adapting the engine to support your new intermediate representation as input.

Research Questions

- Which of the existing regex rewrites found in optimizers or papers are correct for JavaScript regexes?
- How do we decide which rewrites to use in our regex optimizer to ensure that the new regex is faster to match?
- Is our contextual equivalence criterion complete?
- What intermediate representation of regexes can both allow more optimizations and serve as an input to a regex engine?

Prerequisites

- Experience with the Rocq proof assistant.
- Interest in formal semantics and formal verification.
- Writing (depending on progress, we could aim to submit this work to a conference, like POPL or CPP).

Get in touch with `aurele.barriere@epfl.ch`, `clement.pit-claudel@epfl.ch` and `gabriel.radanne@inria.fr` if you are interested!

References

[1] Barrière, A., and Pit-Claudel, C. Linear matching of javascript regular expressions. *Proc. ACM Program. Lang. 8*, PLDI (2024), 1336–1360.

[2] Barrière, A., Deng, V., and Pit-Claudel, C. Formal verification for javascript regular expressions: a proven semantics and its applications, 2025.

[3] Chida, N., and Terauchi, T. Repairing regular expressions for extraction. *Proceedings of the ACM on Programming Languages 7*, PLDI (2023), 1633–1656.

[4] De Santo, N., Barrière, A., and Pit-Claudel, C. A coq mechanization of javascript regular expression semantics. *Proc. ACM Program. Lang. 8*, ICFP (2024), 1003–1031.

[5] Dominus, M. J. Perl regular expression matching is NP-hard. https://perl.plover.com/NPC/NPC-3SAT.html, 2000.

[6] ECMA. Ecma-262, 14th edition: Ecmascript® 2023 language specification, June 2023.

[7] Freydenberger, D. Extended Regular Expressions: Succinctness and Decidability. In *Leibniz International Proceedings in Informatics (LIPIcs) series* (Dortmund, Germany, Mar. 2011), vol. 9, pp. 507–518.

[8] Loring, B., Mitchell, D., and Kinder, J. Sound regular expression semantics for dynamic symbolic execution of JavaScript. In *PLDI 2019* (2019), K. S. McKinley and K. Fisher, Eds., ACM, pp. 425–438.

[9] Owens, S., Reppy, J., and Turon, A. Regular-expression derivatives re-examined. *Journal of Functional Programming 19*, 2 (2009), 173–190.

[10] Soshnikov, D. `regexp-tree`. https://github.com/DmitrySoshnikov/regexp-tree, 2025.