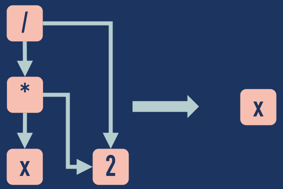
Optimising Rewrite Rulesets

How can an optimised ruleset improve the efficiency of a term-rewriting optimiser?

TU Delft

1. Background Information



Rewriting expressions into simpler and more optimised forms.
Rewrites are facilitated by a set of rules

((?a * ?b) / ?c) ↔ (?a * (?b / ?c))

(?a / ?a) → 1

(?a * 1) → ?a

Rules are matched to sub-expressions to transform them.
Rules that are applied in sequence can be combined into a single rule

((?a * ?b) / ?b) → ?a

By compounding rule chains into a single rule it is possible to shorten the time required for super-optimisation.

2. Process and Methodology

1 Collect a ruleset for C and codebases for experimentation [2]

2 Construct rule-rewriting optimiser for C using EGG[1] [3]

3 Develop methodology for rule-chain extraction [4]

4 Test the performance of base and optimised rulesets [5]

5 Repeat step 4 on varying codebases [6]

6 Analyse the collected results [7]

Methodologies were considered for rule extraction:

1. Naïve method, any chain of rules longer than 2 that affects the expression cost is combined.
2. Common extraction method, apply naïve and only considers chains that occur more than once.

The measure of performance for a ruleset on a given expression is the lowest amount of time that is required by the optimiser to attain the lowest possible cost expression.

[1] M. Willsey, C. Nandi, Y. R. Wang, O. Flatt, Z. Tatlock, and P. Panchekha, "egg: Fastand extensible equality saturation," Proc. ACM Program. Lang., vol. 5, no. POPL, Jan.2021. [Online]. Available: https://doi.org/10.1145/3434304
[2] "Search Problem" icon by SURA DADI, from https://thenounproject.com/icon/search-problem-6858495/ CC BY 3.0
[3] "Build" icon by ahmadwil, from https://thenounproject.com/icon/build-5643781/ CC BY 3.0
[4] "Logistic" icon by Baristalcon, from https://thenounproject.com/icon/logistic-3811073/ CC BY 3.0
[5] "Experiment" icon by mbomboro, from https://thenounproject.com/icon/experiment-6629324/ CC BY 3.0
[6] "Repeat" icon by Paonkz, from https://thenounproject.com/icon/repeat-6245509/ CC BY 3.0
[7] "Evaluate" icon by WARHAMMER, from https://thenounproject.com/icon/evaluate-5772783/ CC BY 3.0

3. Experiments and Results

Described experiment was conducted on the following codebases:

- Set of competitive programming solutions
- Synthetically generated
- GZIP source code

- Each experiment has demonstrated increases in performance ranging from 1.6 to 2 times increase. Results for the GZIP codebase can be seen in Figure 1.
- No common rules were extracted from the competitive solutions codebase.
- Optimised rulesets produce decreases in performance when applied to codebases they were not optimised on – poor generality.

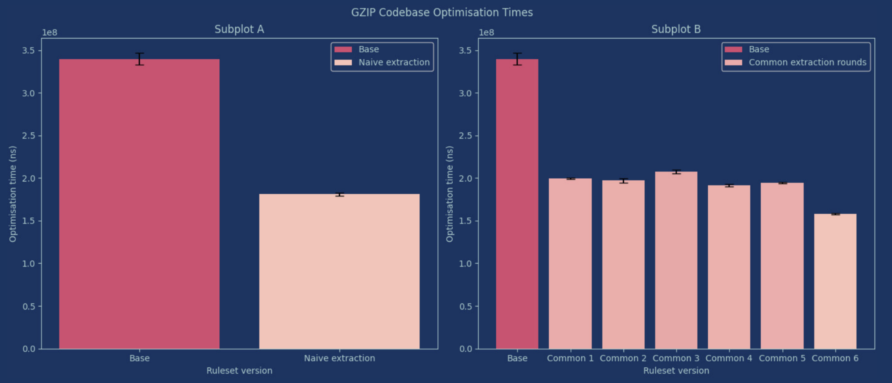


Fig 1. Comparison of required optimisation times for base rulesets, naively optimised(Subplot A) and common extraction (Subplot B)

4. Conclusions

- Both techniques have demonstrated capacity to increase optimiser performance when applied to ideal conditions and real-world scenarios.
- Common extraction method performed better than naïve method. This is due to the overhead introduced by a larger ruleset outweighing performance improvements (Figure 1, Subplot B).
- Ruleset optimisations result in over-specialisation of rulesets on a given codebase as suggested by the optimised rulesets exhibiting poor generality.

5. Limitations and Future Research

- Implementing lossless transpilation for more accurate results and verification of correctness of optimised expressions.
- Devise and investigate more sophisticated rule chain extraction and selection techniques.
- Investigate utility of the techniques when applied to a more diverse set of codebases.
- Research applicability of naïve and common extraction techniques for languages representing other programming paradigms.

Author: Mark Ardman
Contact information: M.A.Ardman@student.tudelft.nl

Supervisor: Dennis Sprokholt
Responsible professor: Soham Chakraborty