Reproducing the Evaluations for *Chisel* VS *Perses*

Funmilayo Olaiya & Gareema Ranjan

Cheriton School of Computer Science

University of Waterloo

DEC. 2. 2022

Software Testing & Debugging

Program Reduction

Software Debloating

Introduction & Background & Motivation

Software Debloating

Problem Statement

- Inaccurate and questionable depiction of Chisel's performance against Perses which leads to two research questions.

RQ1

How much does reinforcement learning help Chisel in program reduction?

RQ2

How effective is Chisel as compared to Perses in C program reduction?

Major Related Work

Chisel

1 Effective Program Debloating via Reinforcement Learning By **Heo et al.**

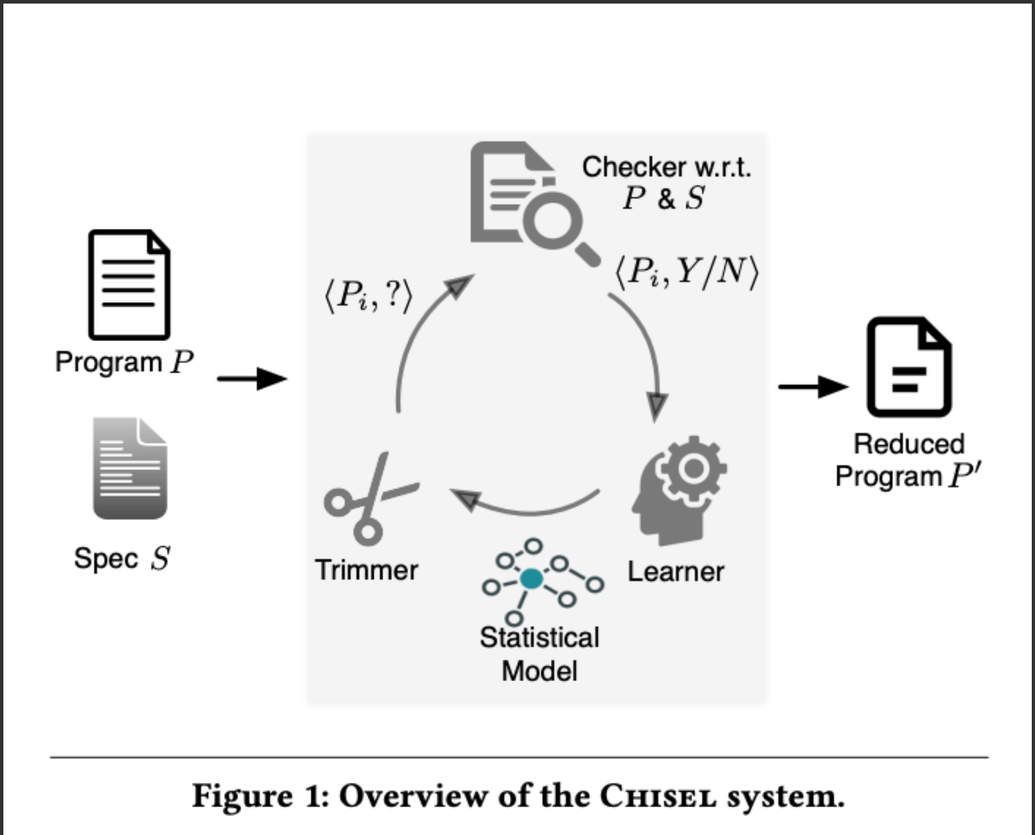


Figure 1: Overview of the CHISEL system.

- Claims other state-of-the-art tools like C-Reduce & Perses does not satisfy their introduced criteria (MERNG)
- Claims Perses sacrifices Efficiency and Generality.
- Proposes a reinforcement learning framework for efficient and scalable program reduction

Chisel as a "custom and debloating" tool

Original program

```c
int printf(const char *p, ...);

int main(int argc, char *argv[])
{
    char *op = argv[2];
    int first = atoi(argv[1]);
    int second = atoi(argv[3]);
    switch (*op)
    {
    case '-':
        printf("%d - %d = %d\n", first, second, first -
second);break;
    case '+':
        printf("%d + %d = %d\n", first, second, first +
second);break;
    case '*':
        printf("%d * %d = %d\n", first, second, first *
second);break;
    case '/':
        printf("%d / %d = %d\n", first, second, first /
second);break;
    default:
        printf("Error! operator is not correct");
    }
    return 0;
}
```

Sample Chisel Test Script

```
#!/bin/bash

export BENCHMARK_NAME=mycode
export BENCHMARK_DIR=
       $CHISEL_BENCHMARK_HOME/benchmark/$BENCHMARK_NAME/merged
export SRC=$BENCHMARK_DIR/$BENCHMARK_NAME.c
export ORIGIN_BIN=$BENCHMARK_DIR/$BENCHMARK_NAME
export REDUCED_BIN=$BENCHMARK_DIR/$BENCHMARK_NAME.reduced
export TIMEOUT="-k 0.5 0.5"
export LOG=$BENCHMARK_DIR/log.txt

source $CHISEL_BENCHMARK_HOME/benchmark/test-base.sh


function desired() {
rm -rf out1
clang t.c -o out1
./out1 100 + 100 > temp1.txt
readonly EXIT_CODE="$?"
echo $EXIT_CODE

if [[ "${EXIT_CODE}" == "0" ]] && grep -q "200" temp1.txt ;
then
return 0
fi

return 1
}

desired
```

Reduced Program

```c
int printf(const char *p, ...);

int main(int argc, char *argv[]) {
  char *op = argv[2];
  int first = atoi(argv[1]);
  int second = atoi(argv[3]);
  switch (*op) {
   case '+':
    printf("%d + %d = %d\n", first, second,
           first + second);
    break;
  }
  return 0;
}
```

Perses

2

Perses: Syntax-Guided Program Reduction By **Sun et al.**

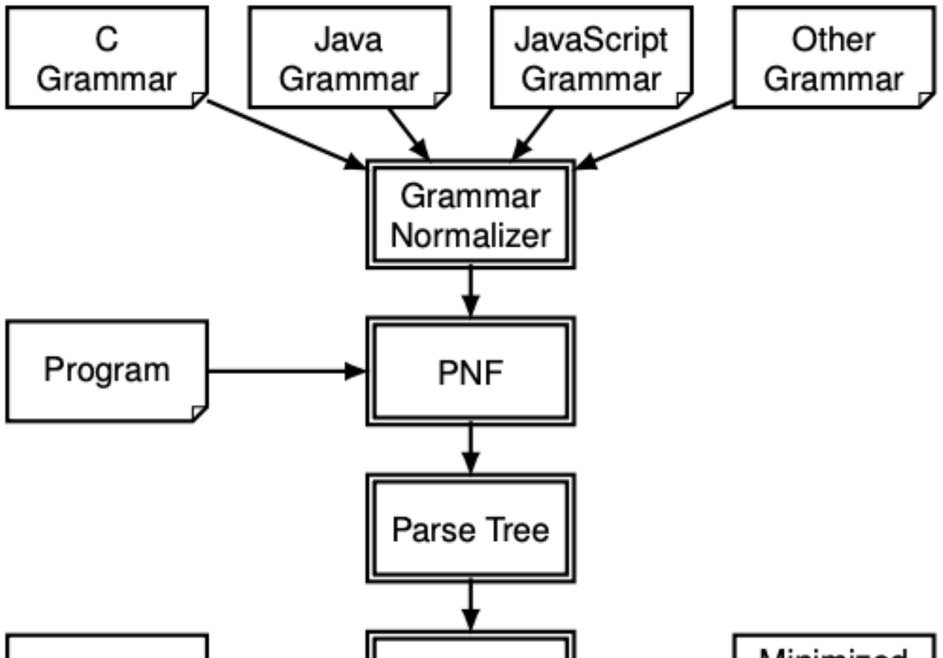


Figure 3: Overall Workflow of Perses.

- Effective program reduction using a syntax-guided technique
- No need for customising functionalities

Methodology

Evaluation Approach

1. Analyzing the Chisel codebase especially in terms of RL
2. Running some of Chisel's and Perses's benchmarks
3. Creating our own benchmarks
4. Analyzing the different use cases of both Chisel and Perses

Addressing RQ1

On CHISEL'S Reinforcement Learning

Evaluating **Chisel** as a program debloating tool *via Reinforcement Learning*

```
#include "ProbabilisticModel.h"

#include <fstream>
#include <sstream>
#include <vector>

#include <mlpack/core.hpp>
#include
<mlpack/methods/decision_tree/decision_tree.hpp>
#include "OptionManager.h"
#include "Profiler.h"
```

https://www.mlpack.org/gsocblog/deep-reinforcement-learning-methods-summary.html

```
void ProbabilisticModel::train(int Iteration) {
    if (OptionManager::SkipLearning)
       return;
     Profiler::GetInstance()->beginLearning();
     bool ShouldTrain =
        !(Iteration > 100 && Iteration % (Iteration / 100 + 1) !=
0);  if ((!OptionManager::SkipDelayLearning && ShouldTrain) ||
         OptionManager::SkipDelayLearning) {
      MyDecisionTree.Train(TrainingSet, TrainingLabels, 2, 1);
      }
     Profiler::GetInstance()->endLearning();
}
```

Addressing RQ2

On CHISEL'S Criteria

RQ2 was geared toward the set of criteria stated in the research paper that introduced Chisel

1. **Minimality**: Does the system trim code as aggressively as possible while respecting the specification?
2. Efficiency: Does the system efficiently find the minimized program and does it scale to large programs?
3. Robustness: Does the system avoid introducing new errors and vulnerabilities in the generated program?
4. Naturalness: Does the system produce debloated code that is maintainable and extensible?
5. Generality: Does the system handle a wide variety of different kinds of programs and specifications?

Chisel vs Perses Reduced Programs

```c
int printf(const char *p, ...);

int main(int argc, char *argv[]) {
  char *op = argv[2];
  int first = atoi(argv[1]);
  int second = atoi(argv[3]);
  switch (*op) {
   case '+':
    printf("%d + %d = %d\n", first, second,
           first + second);
    break;
  }
  return 0;
}
```

Reduced program for chisel

```c
    main(int argc, char *argv[])
{
    int first = atoi(argv[1]);
    int second = atoi(argv[3]);
        printf("%d + %d = %d\n"                    , first +
$econd);
```

Reduced program for perses

Tables showing the data gotten from running the benchmarks

Chisel with RL

Benchmarks	Original Size	Reduced Size	Success/Failure	Time -Run1	Time -Run2	Time -Run3
calculator	31	18	YES	1.1	1.1	1
compare-strings	28	15	YES	1.7	1.7	1.7
complex	28	10	YES	1.2	1.2	1.2
decimal-to-binary	29	--	--	TIMEOUT	TIMEOUT	TIMEOUT
decimal-to-binary-alt	143	--	--	TIMEOUT	TIMEOUT	TIMEOUT
fibonacci	45	44	YES	2.3	2	2.1
floatingPointNumber	17	13	YES	0.6	0.5	0.5
palindrome	54	23	YES	1.8	1.7	1.8
random-number-range	30	21	YES	0.6	0.5	0.5
reverse	53	31	YES	5.7	5.8	5.8
simple	15	8	YES	0.8	0.8	0.8
swap_and_sum	46	30	YES	3.3	3.2	3.2

Chisel without RL

Benchmarks	Original Size	Reduced Size	Success/Failure	Time -Run1	Time -Run2	Time -Run3
calculator	31	18	YES	1.1	1.1	1
compare-strings	28	15	YES	1.8	1.8	1.7
complex	28	8	YES	1.3	1.3	1.3
decimal-to-binary	29	--	--	TIMEOUT	TIMEOUT	TIMEOUT
decimal-to-binary-alt	143	--	--	TIMEOUT	TIMEOUT	TIMEOUT
fibonacci	45	44	YES	2.1	2.1	2.1
floatingPointNumber	17	13	YES	0.6	0.5	0.6
palindrome	54	23	YES	1.7	1.7	1.8
random-number-range	30	21	YES	0.6	0.5	0.6
reverse	53	31	YES	5.7	5.7	5.8
simple	15	8	YES	0.9	0.8	0.9
swap_and_sum	46	30	YES	3.3	3.3	3.3

Perses

Benchmarks	Original Size	Reduced Size	Success/Failure	Time -Run1	Time -Run2	Time -Run3
calculator	31	6	YES	15	14	16
compare-strings	28	1	YES	0	0	0
complex	28	3	YES	7	7	7
decimal-to-binary	29	--	--	TIMEOUT	TIMEOUT	TIMEOUT
decimal-to-binary-alt	143	--	--	TIMEOUT	TIMEOUT	TIMEOUT
fibonacci	45	13	YES	27	27	26
floatingPointNumber	17	3	YES	1	1	1
palindrome	54	6	YES	20	19	20
random-number-range	30	3	YES	1	1	1
reverse	53	--	--	TIMEOUT	TIMEOUT	TIMEOUT
simple	15	5	YES	4	6	4
swap_and_sum	46	--	--	TIMEOUT	TIMEOUT	TIMEOUT

On Chisel Vs Perses Speed

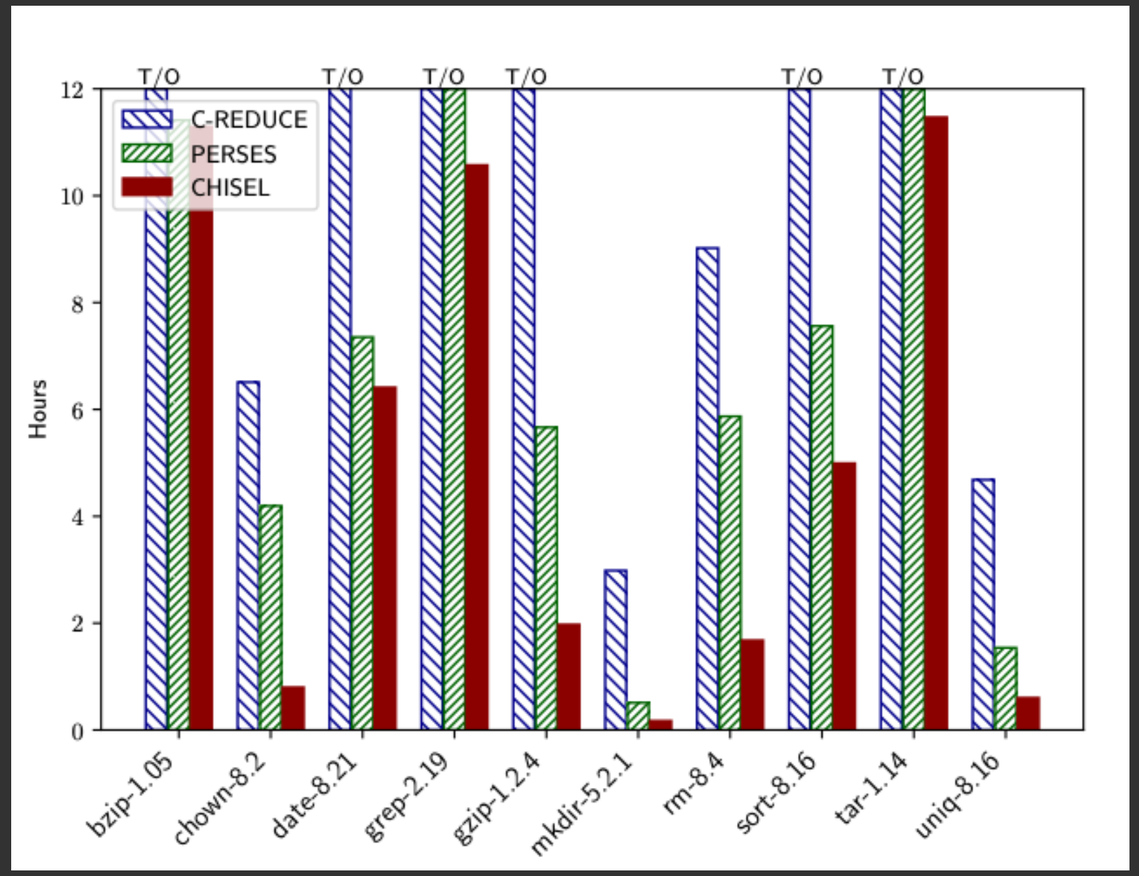
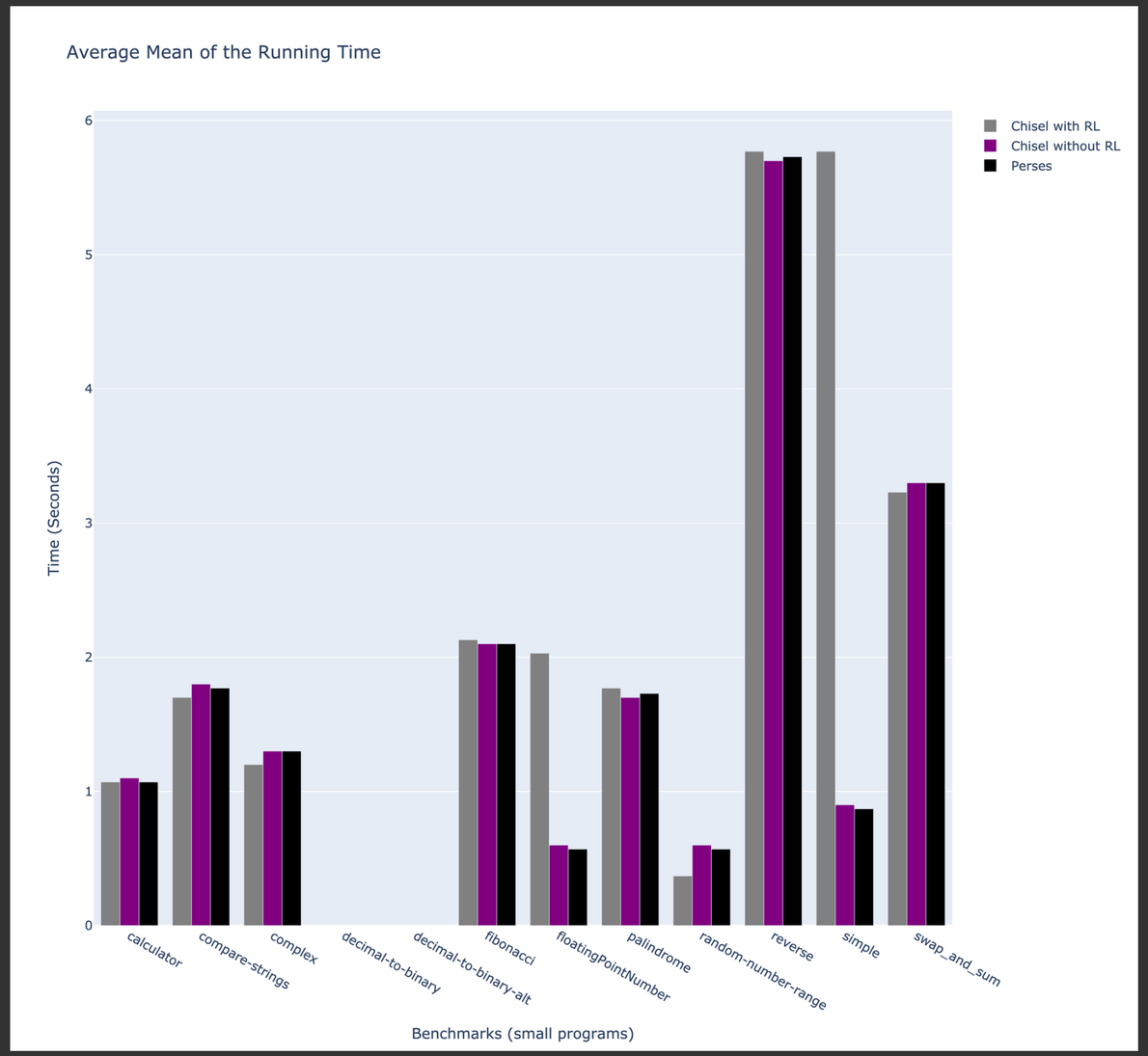
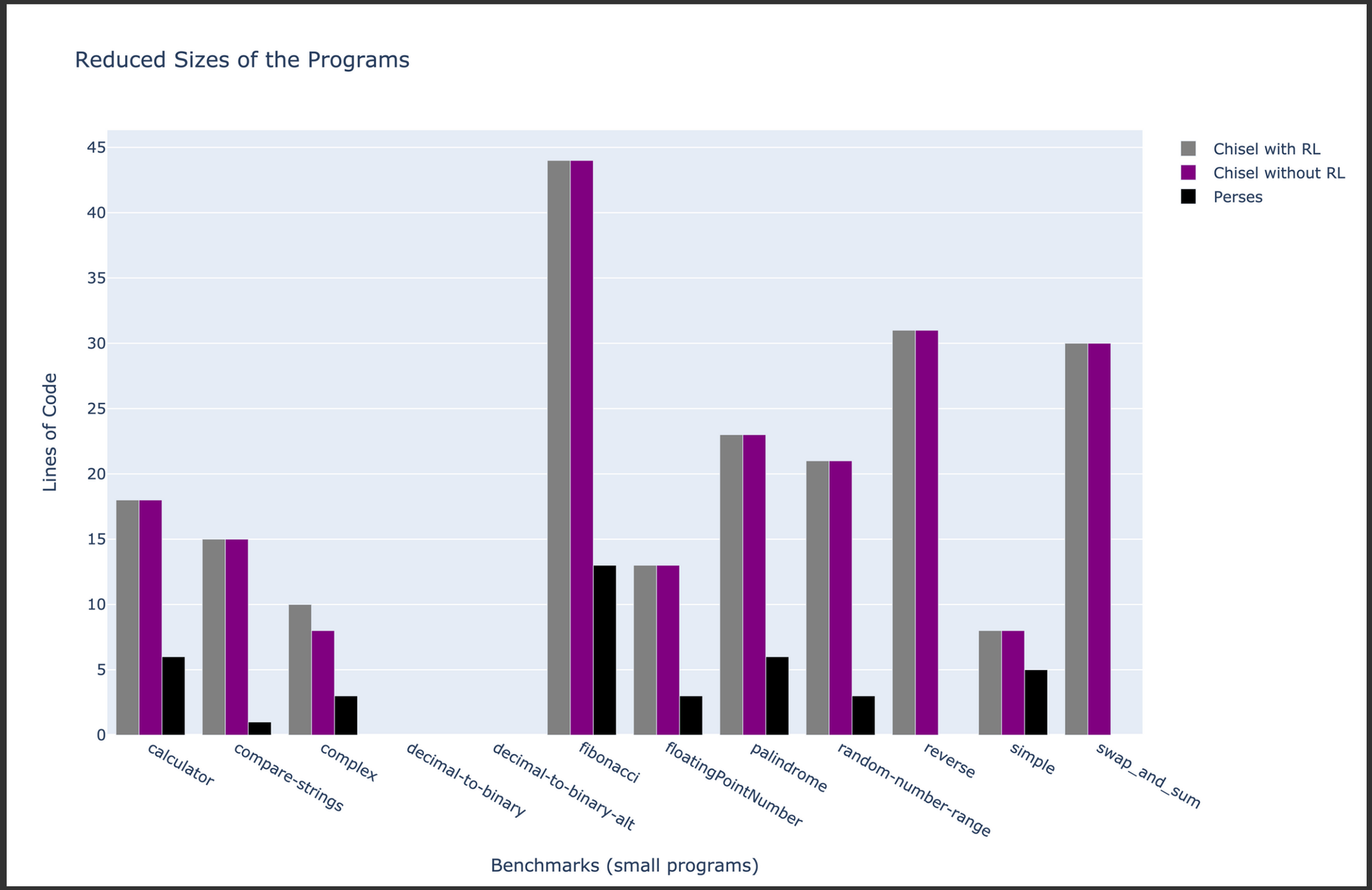


Figure 9: Original chart of evaluations in the paper



On Chisel Vs Perses Program Reduction Size



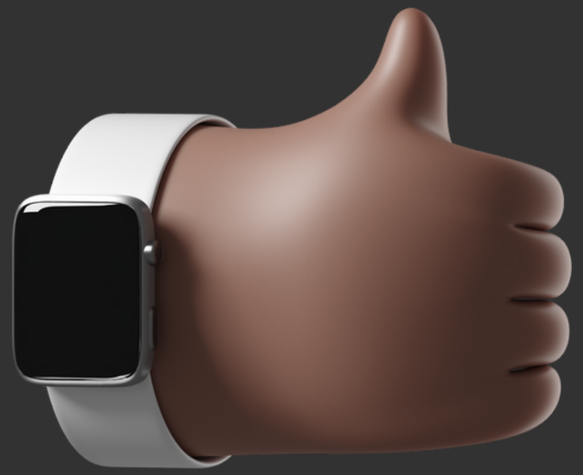
Few Limitations

1. Not having deeper knowlege of RL

2. We couldn't use the benchmarks from both Chisel & Perses

Conclusion

1. One thing we will give to Chisel ~ It's fast, but efficiency is more than just 'speed' and 'timeout'
2. Chisel is a *"customise and debloat"* tool while perses is not
3. The use of RL with Chisel is not significantly better than using Chisel without RL
4. We believe Chisel is a good tool – but a shabby job was done on definite comparisons



Thank you for listening!

Any Questions?