Contents

Is Software Debloating Useful? - A Comparative Study

Sumit Lahiri, Amit Kumar Sharma

What is Program Debloating? What is Program Debloating?

***Index Terms*—Software Debloating, Software Engineering, Delta Debugging**

I. Project Objective

We introduce and explain the issue involving **software bloating** that comes in given the nature and varied options available to develop modern software. We restrict ourselves to `C` or `C++` projects especially the once used frequently as a part of `unix` or `darwin` oses. These projects are usually build and installed as standalone tools or as libraries for linking with other major tools.

We propose to debloat a piece of `C or C++ Code` via **Policy Based Reinforcement Learning** using techniques and approaches as laid out in the two research papers we have selected for accomplishing the task. Both the papers have demonstrated novel techniques for debloating the codebase for a given `C or C++ Software` using **Policy Based Reinforcement Learning**, the first paper mentioned does this by **Learning-Guided Delta Debugging** while the second paper does this by **Guided Function Specialization** technique.

In our project (based out of Paper 2), we model the debloating problem as **reinforcement learning** where action would be to specialize a program or not. The state of the model will be a **vector** containing all the required details about the **specialized code**. The **reward** will be a **reduction** of the number of **instructions or code size** in the program after the program is debloated. As the model keeps learning, we expect the rewards to be improved with more inputs.

Software deblaoting leads to reduction of attack surface and lesser code to build and install, given it is done meticulosuly. We use debloating techniques to perform debloating on these tools or library source

code and report back the reductions we saw after debloating. As a part of the process we get a deeper understanding of each of the tools and how they function, modify them to get comparision metrics or implement them Eg. `DeepOCCAM` from research papers based on available source code.

At the end we have a comprehensive report and working implementation of the modifiactions or runs we do to either in the benchamrks or in the tool's `source code` to suite the needs of the project. Given the limited time we got, not all things were completely implemented, especially the runs on other common benchmarks would be needed for a clearer comparison of the tools. Nevertheless, we provide a comparison report based on the current runs and share our insghts on the suitability of use of each tool in different contexts `(What technique works better than the other?)`

II. What is Software debloating?

Software bloating is quite a common problem in any real world software project where the code base is plagued with LOCs that are not useful while the program runs in a given execution context, a common example being exposing too many **redundant APIs**, **default configurations** for each context or many **command line options** that are not used or never invoked in general but are still in the codebase. One primary reason for this is that it isn't possible to structure the code base before hand or to choose a strict design pattern for all components that we write. Many times code needs to be written on demand or ad-hoc basis because not all requirements are captured in the early stages of the project and that becomes the potential root cause for **software bloating**. **Software bloating** can lead to bugs, slower code execution and even expose vulnerabilities in the code base. The current techniques used are **manually thought** clever **metrics & heuristics** for identifying such **bloat sites** and refactoring the code to remove them.

A. Software Debloating Tools

We focus on four popular tools we found for the purpose of software debloating of large scale `C or C++` projects, namely `Chisel`, `Trimmer`, `OCCAM` & `DeepOCCAM` tools. These tools vary in the way they do debloating and the specifications needed in order to do effective and sound debloating. By effective and sound deblaoting, we are referring to thier capacity to reduce `function calls`, `instructions`, `direct loops` etc from the original source code and produce a binary that is best suited to a given runtime environment context with out hampering the normal `desired` execution of the program. On different `environemnts`, based on the specifications we give and the way the binary is executed, the exact debloating effect of `reductions` may be differ significantly since the `desired` property specification may vary based on the `OSes` we run them on or the `settings/parameters` needed for normal run.

In each of these tools, we need to provide some information in the form of `desired` properties that we want our final program to meet under all safe condition and remove the other parts of the `code` or `instructions` that we dont need in the current execution context of the final binary produced.

B. Software Debloating Techniques

The techniques used by these tools for effective debloating varies from tool to tool but the the overall nature of the transformation can be catagorized into two types `source-to-source` & `source-to-binary` transformation.

In `source-to-source` transformation the tool's input is the bloated `source code` and the `specification` in the form of tests that need to pass under a given execution environment. The tool produces a `debloated` source code and binary from the input **source**. This is usually guided by a machine learning model and the reduction happens via techniques similar to `dead code elimination`. Some examples of tools running like this are `Chisel` & `Razor`

In `source-to-binary` transformation the tool's input is the bloated `source code` and the `specification` listing statically known arguments to the `main()` function and other `dynamic arguments` needed under a given execution environment. The tool produces a `debloated` binary from the input **source** but doesnot produce `debloated source code` unlike the technique mentioned above.

This is usually guided by hand crafted heuristics for `function specialization` at each `call-site` and the reduction happens via techniques similar to `dead code elimination` or other `compiler optimization` passes. Some examples of tools running like this are `Trimmer` & `OCCAM` but the exact algorithms for `constant propagation`, `function specialization` & `function inlining` differs in both the tools.

Another approach for `function specialization` at each `call-site` is by using machine learning to decide as to `when to specialize` and then do regular reduction in a similar fashion as above. `DeepOCCAM` is an example of this technique.

III. Motivating Examples

As a motivating example we show what current `compiler optimization` techniques fail to do and **why** special software deblaoting tools are needed for the purpose of `debloating`, we here show the working of the `Chisel` tool since `source-to-source` transforamtion is of `particular interest` to most software developers. We show the `original` source code

```
#include <stdio.h>
void run(int a) {
  if (a > 90) {
    printf("%d\n", a);
  }
}
long long int add(int a, int b)
{ return a + b; }
long long int sub(int a, int b)
{ return a - b; }
int main(int argc, char *argv[]) {
  int c = 0;
  c = -500;
  if (c > 0) {
    run(c);
    add(c, c + 1);
  } else {
    sub(c + 90, c);
  }
  return 0;
}
```

and the `chiseled` source code below it on a blank test case where nothing is `desirable` in the code.

```
#include <stdio.h>
int main(int argc, char *argv[]) {
  int c = 0;
  return 0;
}
```

We were amazed at the amount by which `Chisel` was able to reduce the source code via `reinforcement guided delta-debugging` on this blank test case. We now show what `gcc` with `-O3` optimization was able to do with the code. This is acceptable since the **gcc compiler** has no way to remove the code directly based on current techniques of `code optimization` and thus `debloaters` to the rescue.

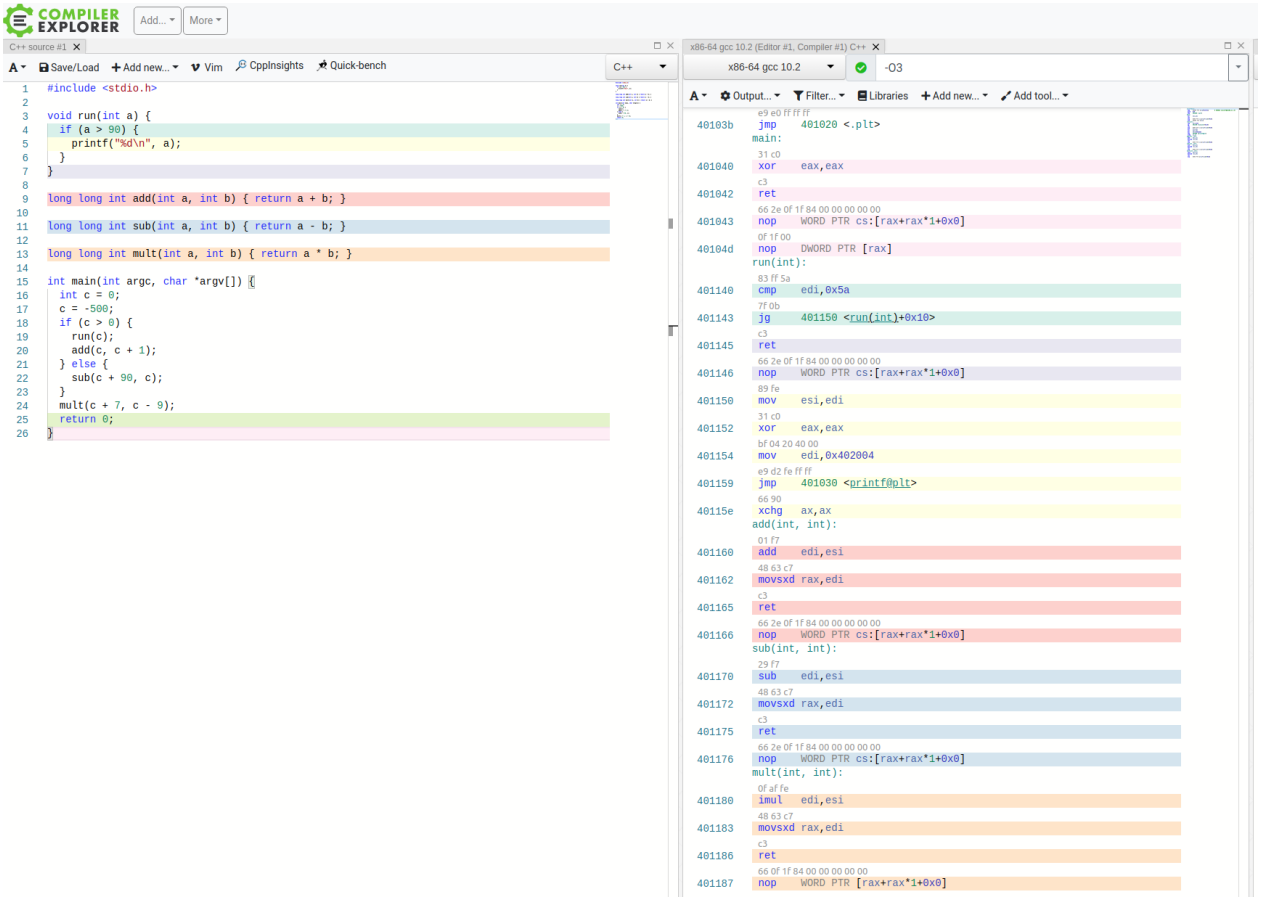

Figure 1: **C Code** Example before **Chisel** tool processing on **blank test** case

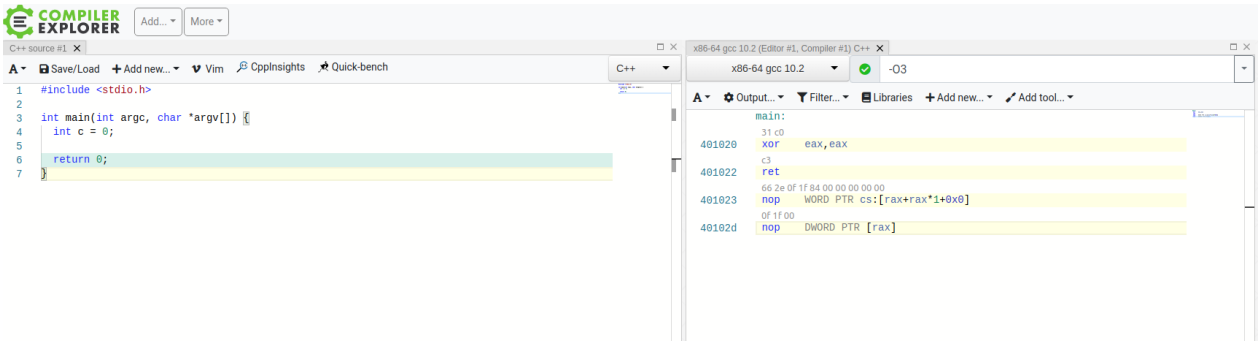

Figure 2: **C Code** Example after **Chisel** tool processing on **blank test** case

We now show **Chisel** tool in action.

```
Start global reduction
Running delta debugging - Size: 3
Start local reduction
Reduce process_aflag at test.c
Running delta debugging - Size: 1
Reduced - Size: 0
Reduce process_bflag at test.c
Running delta debugging - Size: 2
Reduced - Size: 1
Reduced - Size: 0
Reduce process_cflag at test.c
Running delta debugging - Size: 3
Reduced - Size: 1
Reduced - Size: 0
Reduce main at test.c
```

```
Running delta debugging - Size: 2
Reduced - Size: 1
Iteration 2 (Word: 58)
Start global reduction
Running delta debugging - Size: 3
Reduced - Size: 1
Reduced - Size: 0
Start local reduction
Reduce main at test.c
Running delta debugging - Size: 1
Iteration 3 (Word: 43)
Start global reduction
Running delta debugging - Size: 0
Start local reduction
Reduce main at test.c
Running delta debugging - Size: 1
Reduce File: test.c
Iteration 1 (Word: 43)
Start global reduction
Running delta debugging - Size: 0
Start local reduction
Reduce main at test.c
Running delta debugging - Size: 1
```

We now show a working of the OCCAM tool on a different `getopt()` based C example. We pass in some default static arguments to the program and see the OCCAM tool in action on it.

We present the `static analysis` details for OCCAM run in before & after all `inter` & `intra` specialization passes have been completed. The example C code is as below. It accepts `three flags` namely `-a`, `-b` & `-c` and call a function based on the `flag` recieved.

```c
#include <stdio.h>
#include <stdlib.h>

void process_aflag(int a)
{ printf("%d\n", a + 90); }

void process_bflag(int a) {
  a = 80 * a;
  process_aflag(a);
}

void process_cflag(int a) {
  a = a << 20;
  process_aflag(a);
  process_bflag(a);
}
// Example based on : http://osr507doc.sco.com/
    en/tools/ccs_stdio_args.html
int main(int argc, char *argv[]) {
  /* Function flags */
  int aflag = 0;
  int bflag = 0;
  int cflag = 0;
  int ch;

while ((ch = getopt(argc, argv, "abc")) != -1)
    {
  /* For options present */
  /* set flag to some value */
  /* else write out an error statement */
  switch (ch) {
    case 'a':
      aflag = 10;
      fprintf(stdout, "aflag set !\n");
      process_aflag(aflag);
      break;
    case 'b':
      bflag = 20;
      fprintf(stdout, "bflag set !\n");
      process_bflag(bflag);
      break;
    case 'c':
      cflag = 30;
      fprintf(stdout, "cflag set !\n");
      process_cflag(cflag);
      break;
    default:
      (void)fprintf(stderr, "Usage: %s [-abc]\n
          ", argv[0]);
      return (2);
    }
  }
  if (aflag < 0) {
    process_cflag(90);
  }
  /* Do other processing controlled by aflag,
      bflag, cflag. */
  process_bflag(60)
  return (0);
}
```

The `Manifest file` we used for running `OCCAM` in `aggressive mode`

```
{ "main" : "test.o.bc"
, "binary"  : "test"
, "modules"    : []
, "native_libs" : []
, "static_args"    : ["-a", "90"]
, "name"    : "test"
}
```

`OCCAM` log after run was complete. We see that there was reduction of the code size that was finally complied to binary.

```
Statistics for  before specialization
[CFG analysis]
4   Number of functions
0   Number of specialized functions
0   Number of bounced functions added by devirt
15  Number of basic blocks
81  Number of instructions
13  Number of direct calls
6   Number of external calls
0   Number of assembly calls
0   Number of indirect calls
0   Number of unknown calls
1   Number of loops
0   Number of bounded loops
[Memory analysis]
37  Number of memory instructions
32  Statically safe memory accesses
5   Statically unknown memory accesses

Statistics for  after specialization
[CFG analysis]
4   Number of functions
0   Number of specialized functions
0   Number of bounced functions added by devirt
15  Number of basic blocks
60  Number of instructions
13  Number of direct calls
7   Number of external calls
0   Number of assembly calls
0   Number of indirect calls
0   Number of unknown calls
1   Number of loops
0   Number of bounded loops
[Memory analysis]
19  Number of memory instructions
15  Statically safe memory accesses
4   Statically unknown memory accesses
```

IV. OCCAM Tool

OCCAM is a debloating tool that works on the principle of winnowing and object culling. Winnowing is a static analysis and code specialization technique based on the partial evaluation algorithm. It is a code optimization technique where all the static inputs computations are processed during compile time. Also, all the function arguments are replaced with constant value (if statically known) and optimization passes by LLVM is applied after that. PE helps to achieve the residual program which runs faster than the original program with reduction in gadgets and instruction count.

A. Overview

We detail out a bit more about the OCCAM tool here. Function inlining only replaces the function call by the function definition but PE takes an extra step and evaluates the function before the execution of program using statically known arguments via constant folding or constant propagation in the function body. Static (compile time) analysis is seperate from dynamic (run time) analysis with this algorithm and thus we present details of the tool runs in two sections Static Analysis and Dynamic Analysis.

Partial evaluation works in two steps: (a) Optimization (b) Specialization In optimization phase compile time constants are identified, dead codes are eliminated and the control flow of the program is reduced. In specialization phase, program is effectively specialized across function boundaries.

OCCAM takes two inputs: a source code in C/C++ and a manifest file in JSON format.

The debloated binary of the original source code produced by the OCCAM tool is used by the Gadget set Analyser to get the gadgets count. The original source code is also compiled by a compiler and feed to GSA tool. Both debloated and original binaries are then compared for reduction in metric counts. Dead code elimination in OCCAM works in five ways:

- Aggressive Non-Recursive DCE.
- Inter Procedural DCE. (accross function calls)
- Intra Procedural DCE. (for basic blocks in function body)
- Sparse Conditional Constant Propagation based DCE. (for Trimmer, we enable this pass)
- Abstract Interpretation based DCE. (Seahorn Crab based DCE pass)

B. Working Procedure

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

V. Trimmer Tool

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

A. Overview

TRIMMER tool is used to specialize the target program with respect to the user defined configuration. The configuration contains the usage context of application. Compiler transformation are included in this tool for good debloating.Inter procedural constant propagation is used in the tool which aggressively removes the unnecessary codes. With the reduction in the unused program codes, the gadget count of the program is also reduced and helps to improve the security performance of the system.

B. Working Procedure

The source code is converted to LLVM IR and given as an input to the trimmer tool along with the manifest file consisting of user defined configuration. The TRIMMER first performs input specialization with respect to the user defined configuration . The second part is loop unrolling . The loop unrolling becomes important step to make interprocedural constant propagation easier. Constant Propagation is the final stage in the tool. The specialized code processed by the TRIMMER is then given as input to the linker . The linker also reads the linker flags from the manifest file and generates a final specialized binary executable file.

VI. Chisel Tool

We share a brief information about the tool. **Chisel** tool works by learning a **policy** for **delta debugging** by **reinforcement learning** which guarantees **1-minimal P*** & $O(|P|^2)$ runtime. The abstraction is that a **markov decision process** is being used to model the **reinforcement learning** problem for meaningful **guidance** to learn the **policy** in a better way. All global declarations, variables, functions etc. are first reduced by the delta-debugging principle as state above and thereafter local variables, loop declarations and arguments to functions are optimized. After both the local and global level reductions are done, **Chisel** invokes a run of the **global level reduction** again and repeats the process continually until the **1-minimal P*** version of the program is found.

A. Overview

Chisel tool's working is based on syntax guided hierarchical delta debugging algorithm. The tool ensures that the reduced program are compilable, core functionalities of source program are preserved and undefined behaviours for non core functionalities does not show up. CHISEL also keeps all the criteria required for the system to work properly in order. The criteria are minimality,naturalness,efficiency,robustness and generality. The probabilistic model is used in CHISEL to accelerate the delta debugging algorithm and Markov Decision Process is used to search a proper policy for learning the machine learning algorithm. Model based Reinforcement algorithm is used to converge to the solution quickly.

B. Working Procedure

The input to the **CHISEL** tool is a C/C++ source file (test.c) which is to be debloated. Along with the source file , we give a specialized script which contains the high level specification of the desired output. The CHISEL tool then generates binary of the source code(test.c.chisel.c). Both the source and debloated program are then compiled by g++ or any other compiler and the binaries generated by the compiler are given to the ROP gadgets or Gadget Set Analyser(GSA) . The ROP gadget outputs the count of the gadgets before and after debloating respectively.

C. State Encoding

Markov Decision Process is used for Delta debugging algorithm. For delta debugging algorithm two things are required and the tuple of these two things defines the state of the model at any time. The first thing is the pair of the program to be tested and second thing is the number of partitions into which a program is broken. The initial state consist the entire program represented as a list into two partitions. The program can be broken inti tokens,identifiers ,statements or even a finer granularity is reached to obtain the minimal state.

D. Delta Debugging

Delta debugging is an algorithm or technique to remove the unnecessary part of the input which is not responsible for test case failure. DD makes testing easier as it divides input into smaller subsets because smaller and simplified testcases are easy to handle. Delta debugging algorithm is used till we reach 1-minimal expression. DD is an iterative algorithm.Markov Decision Process for delta debugging is

deployed to build a statistical model to get 1-minimal solution with lesser number of iterations than delta debugging alone.

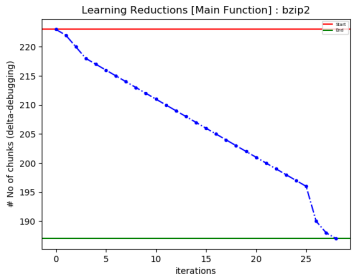

Figure 3: Learning plot for mkdir

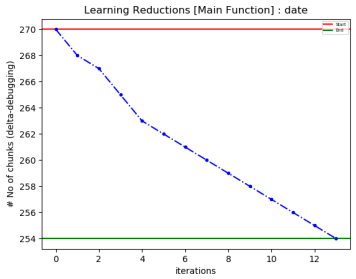

Figure 4: Learning plot for date

E. 1-minimal Program

Figure 5: Learning plot for chown

Suppose we have a testcase T which fails the program P. When T is given as input to P, the entire part of T may not be responsible for causing failure to the program P. Therefore, we are interested in the exact part of T which causes the program to fail. In short, we can say failing test case can have relevant and non-relevant information. To filter out the relevant information from the test case, we use Delta debugging algorithm.If a test case T fails the program P, then the expression T' derived from T is 1-minimal iff any deviation causes the test case failure to go away.

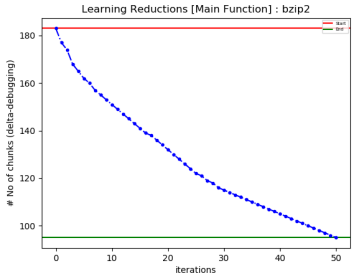

Figure 6: Learning plot for bzip2

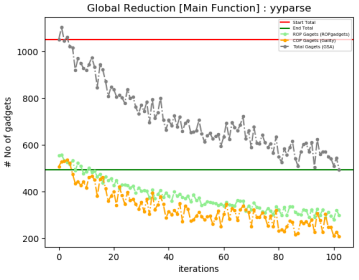

Figure 7: Gadgets plot for yyparse

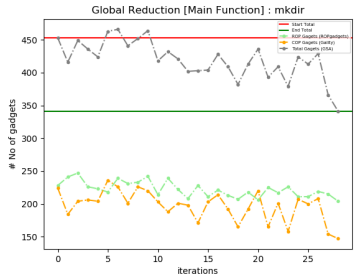

Figure 8: Gadgets plot for mkdir

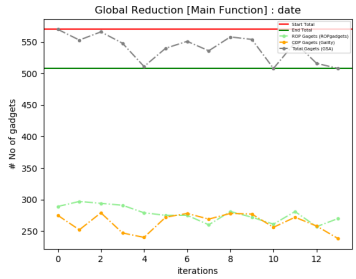

Figure 9: Gadgets plot for date

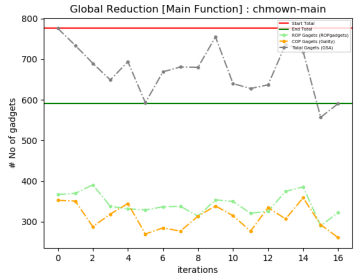

Figure 10: Gadgets plot for chown

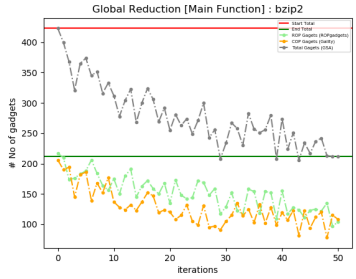

Figure 11: Gadgets plot for bzip2

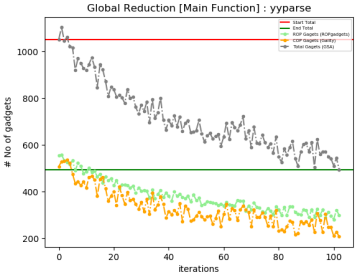

Figure 12: Gadgets plot for yyparse

VII. DeepOCCAM Tool

Deepoccam is an extension of occam tool with new approach based on machine learning. Both occam and Deepoccam are based on the partial evaluation approach. The major contribution of PE is to determine if the function specialization decreases the size of residual program. The function specialization increases the code size of original program but then optimization in the new function such as constant propagation can be possible after function inlining and static values substitution. So, there is an opportunity for code size reduction in the residual program after partial evaluation process.

A. Overview

Whether or not the function specialization reduces the code size is not trivial. This is the problem with occam as it implements ”never specialize” or ”always specialize” approaches only. So, it becomes important that we come up an automatied approach to learn when to specialize. The machine learning model can be considered as a better approach which has been implemented in DeepOCCAM. The specialization depends on the present state of the program and does not consider the past changes made to the program. It follows Markov Decision process. Reinforcement Learning seems to a best fit for this approach where the rewards are reduction in metrics value under consideration. So, DeepOCCAM is implemented with policy-learning based RL model where the rewards are dynamically evaluated and the actions depend on the current encoded state and the metric values.

B. Working Procedure

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Pro-

gram deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

C. State Encoding

State Encoding: Deepoccam have been implemented with two state models: 1. Handcrafted features 2.LLVM IR embeddings Inst2vec

In HF, the state contains the details about the caller,callee, compilation unit and call site. The state in this case is a combination of these four features vectors.

In Inst2vec each instruction is converted to a vector using skip gram model . The LLVM IR of the caller,calle and calling context is represented as a list of instructions. Also the arguments at the call site is represented as a bit vector of 0 & 1 where Zero(0) denotes unknown arguments and one(1) denotes the statically known arguments. This bitvector alongf with the above three vectors form a tuple of the four 2-D matrices. These tuple represents the state for the RL model.

D. Inst2Vec Usage

Inst2vec is the technique of processing the source code into the features vector for modeling the Reinforcement Learning. It follows an approach similar to skip-gram model in Natural Language Processing of pre-defined context size. The source code is first compiled into LLVM IR code. The LLVM IR is in the form of a static single assignment.As LLVM IR is independent of machine or hardware architecture and programming language , it becomes easy to train the program embeddings.The approach is similar to word2vec model.

With the LLVM IR , the contextual flow graph (XFG)is created. XFG takes into account both the data flow and control flow of the program. With these XFGs generated from LLVM IR , the consecutive statement pairs are made of pre-defined context size . These statement pairs are then checked for duplication. The XFGs pairs are made by constructing a dual graph with statement as nodes and removing duplicate edges. The process is then followed by removing statements of negligible presence. We get an inst2vec after subsampling of frequent pairs which can be optimized and trained. XFGs ensures that the semantics of statements are preserved.

VIII. Project Aim

As a experimental & comparison based project, our task was to run the tools on various examples to see for `static` & `dynamic` metric changes **before** & **after** the run of the tools on these examples. We setup up each tools and prepare them for metrics that we want to extract & compare in each of the runs using either statistics that are shown by the tool itself using options like `--stats` or `--opt-stats` for static analysis case or generating intermediate `binary` or `elf-section object` files for dynamic analysis.

We finally report an **overview**, **setup**, **benchmarks** & **comparision** of these tools on some same examples and also on a set of `benchmarks` used originally to test run each of the tools in the original papers.

We also implement a modification of `OCCAM` tool to run like `Trimmer` tool as per description and our understanding of the **Trimmer** original paper and a modification of base `OCCAM` tool to `DeepOCCAM` which uses a `reinforcement learning` based approach to specialization based on **DeepOCCAM** paper.

IX. Implementation Overview

We elaborate on the changes and the work we did to meet the `Project Aim`. We start with the `Chisel` tool where we did modifications to dump the reduced chunk and the original chunk sizes that are used for `markov decision` based `delta-debugging` and used it as a metric to measure the `learning rate` along with reduction in `gadgets` count for the intermediate binary produced by `Chisel` tool

For **OCCAM** tool the modification was to fix the code for running in our environment and then work on the `manifest` & `automated` make builds so that it can be run in `dockers` for final release in the project. Another modification was to dump the `counts` for `caller`, `callee`, `constant arguments`, `statically known arguments` and other `context` related details for `metadata` & `dataset` generation in the base `occam.log` file itself since **we were not able to complete the `gRPC` implementation on time** owing to build and linking related issues.

For **DeepOCCAM**, we found a **half-implemented code** repository which belongs to one of the authors for **DeepOCCAM** paper. The tool doesn't link, build or run on the current platform that we are using. It gave us insights on implementing some of the parts in the code that we are developing as an extension on **OCCAM** to develop **DeepOCCAM**.

The benchmarks we used needed modification interms of running it against updated **gllvm**, **wllvm** & **llvm-10**. There were a few other tools and frameworks that we had to install and test in-order to run the debloating tools on these benchamarks.

We compiled, build & ran the tools in `docker` containers interacting with the base `OS` via `docker volumes` & `u-tty terminal` program.

X. Pipeline Setup

We divided the task of running the `benchmark` & `motivating` examples for the `three` tools into `three` different `pipelines`. Each `pipeline` was built and deployed against different `Ubuntu OS` base `images` which was specifically required for `each` of the tools. There were some `libraries`, `python packages` and linking `object files` that conflicted when trying to run all three tools in the current base `OS` on which we were developing the tools, so we switched to using three `docker` containers one for each `tool`

We ran multiple instances of the `docker` containers for `training` and `parellel` benchmark runs so that we can cover as many as `benchmark` code examples possible. We now show `architecture` diagrams of the final `pipelines` that we intend to use for `demonstration` purposes.

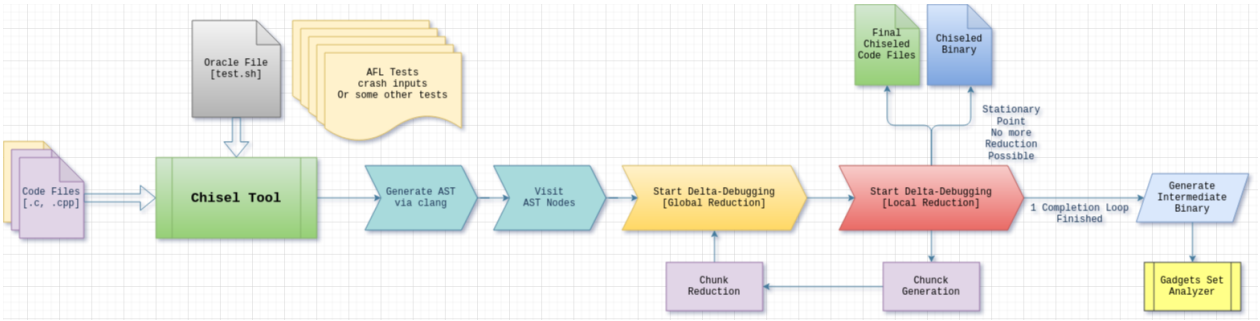

Figure 13: **Chisel Pipeline** : Setup `Chisel` runs on `Chisel-Benchmarks` & other examples

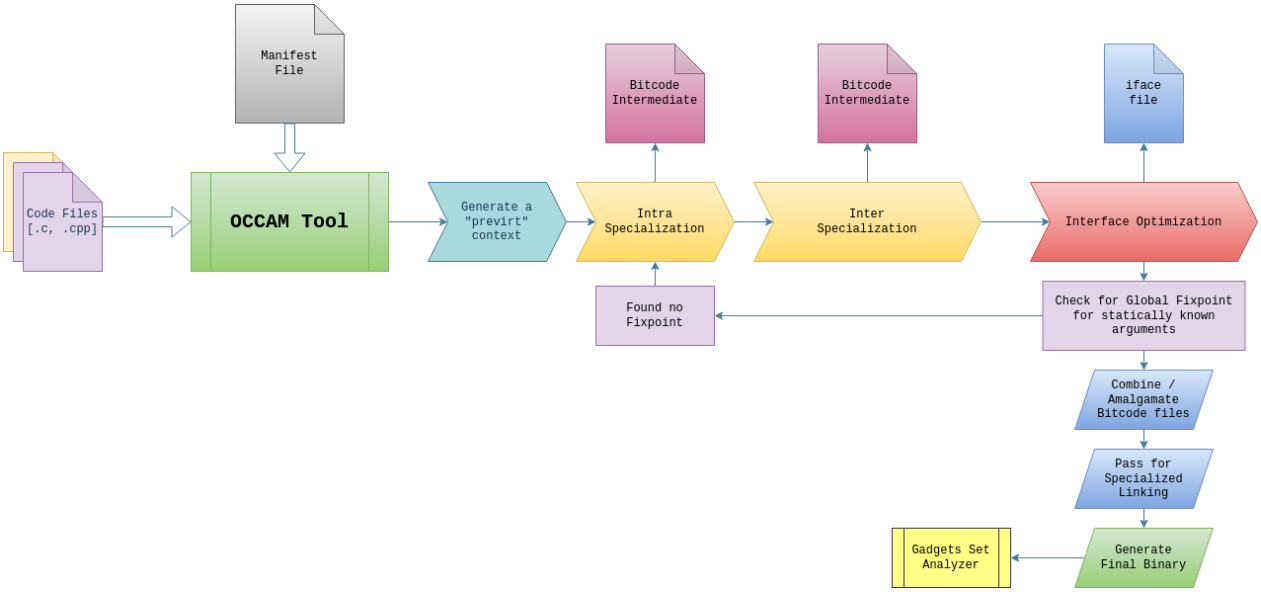

Figure 14: **OCCAM Pipeline** : Setup `OCCAM` runs on `OCCAM-Benchmarks` & other examples

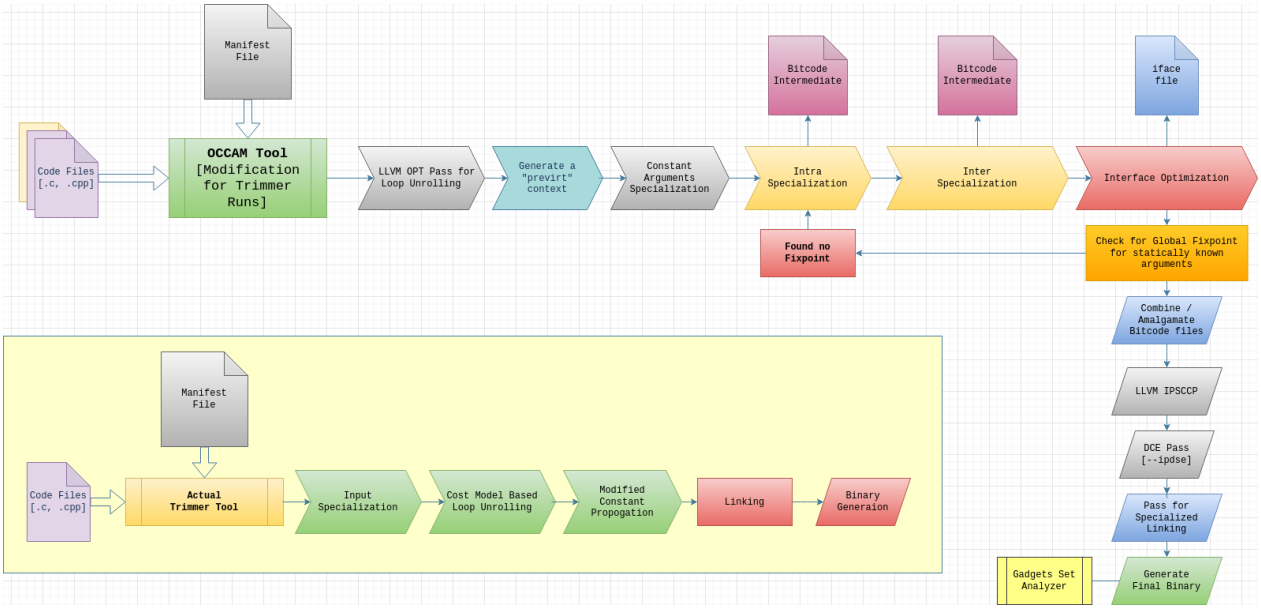

Figure 15: **Trimmer Runs** : Setup `Trimmer` runs on `OCCAM-Benchmarks` only

A. Chisel Pipeline

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

B. OCCAM Pipeline

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

C. Trimmer Runs

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

D. DeepOCCAM Pipeline

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

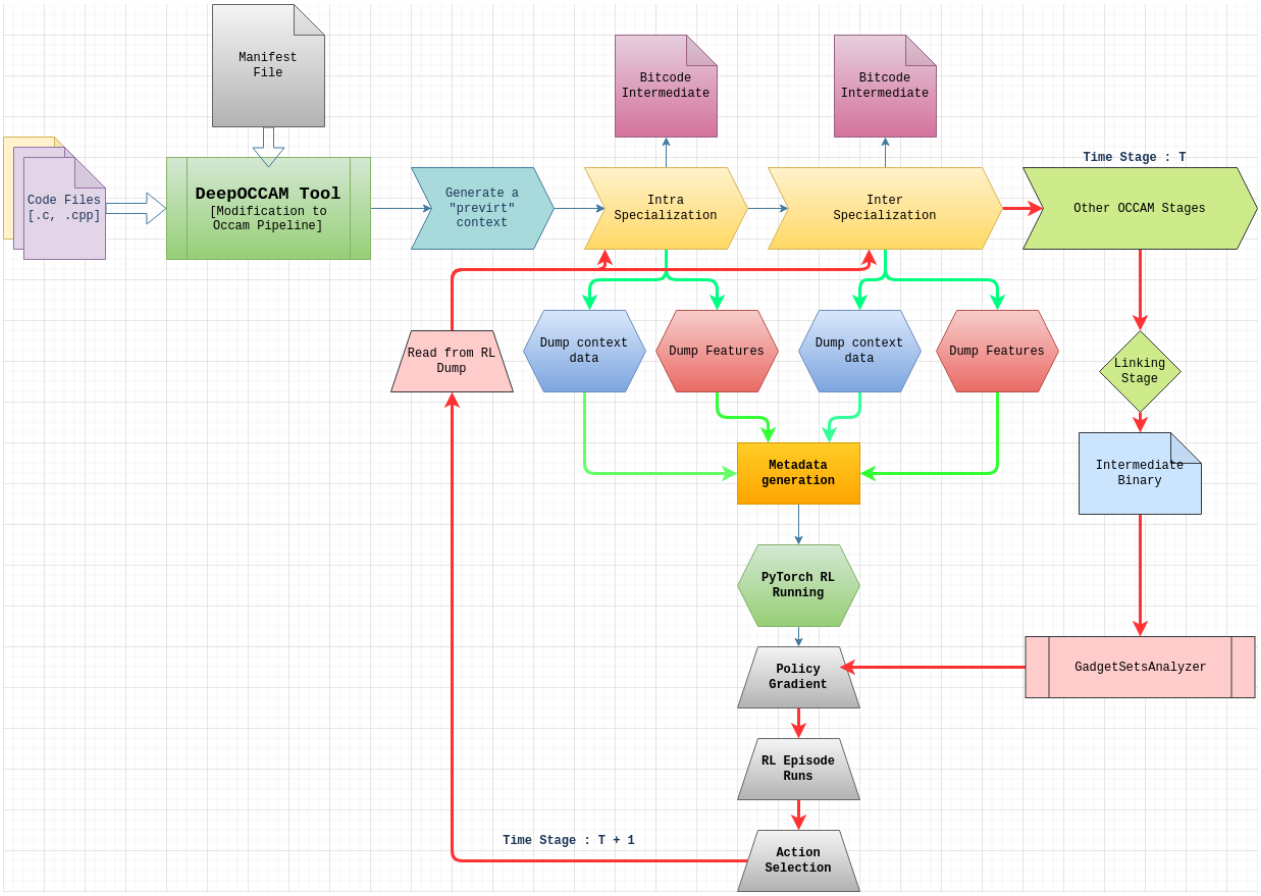

Figure 16: **DeepOCCAM Pipeline**

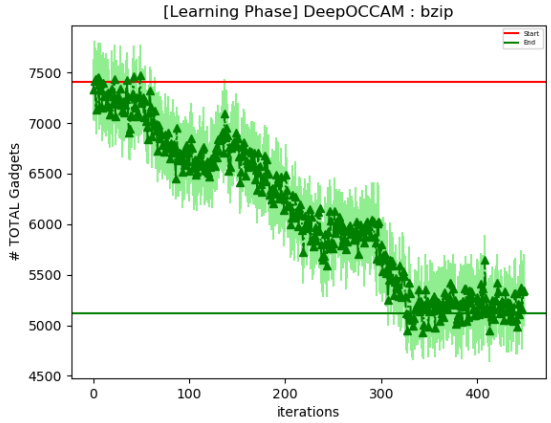

Figure 17: **DeepOCCAM Total**

450 iterations bzip - HF

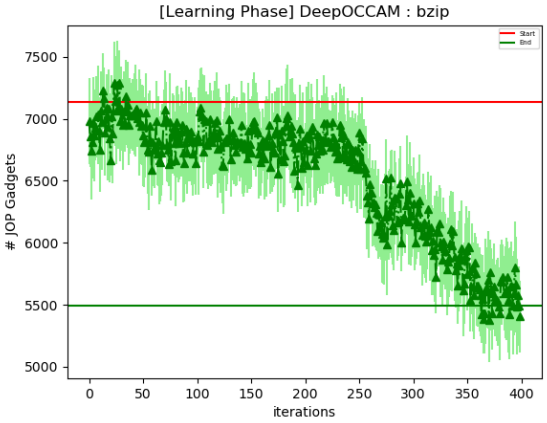

Figure 20: **DeepOCCAM JOP**

400 iterations bzip - HF

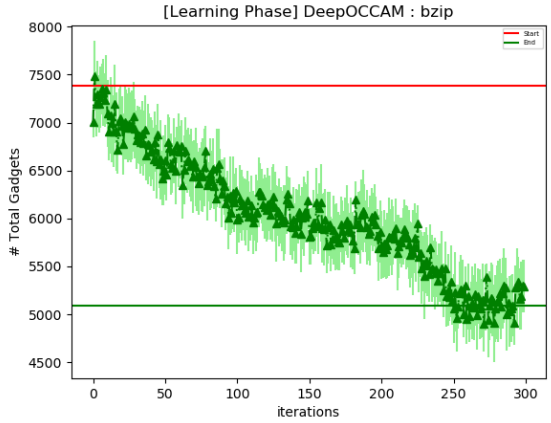

Figure 18: **DeepOCCAM Total**

300 iterations bzip - inst2vec bzip embedding

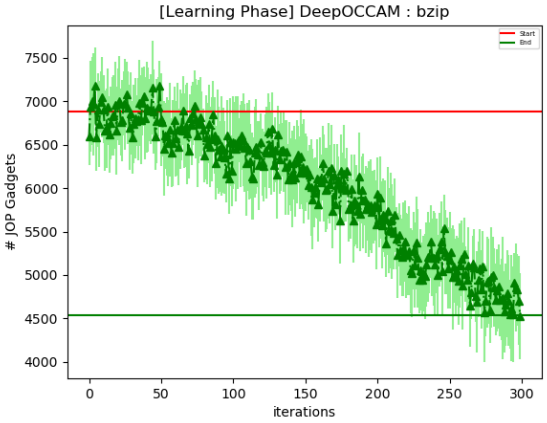

Figure 21: **DeepOCCAM JOP**

300 iterations bzip - inst2vec bzip embedding

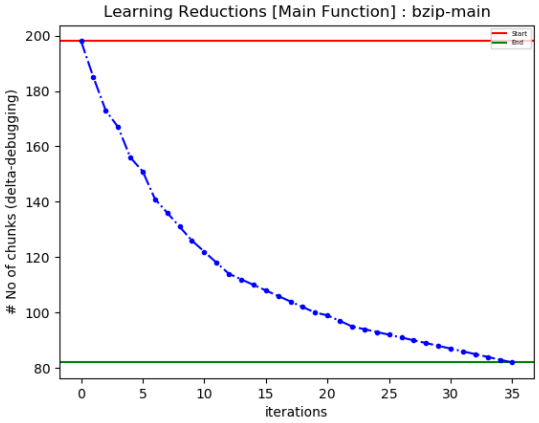

Figure 19: **Chisel** Learning bzip `main()`

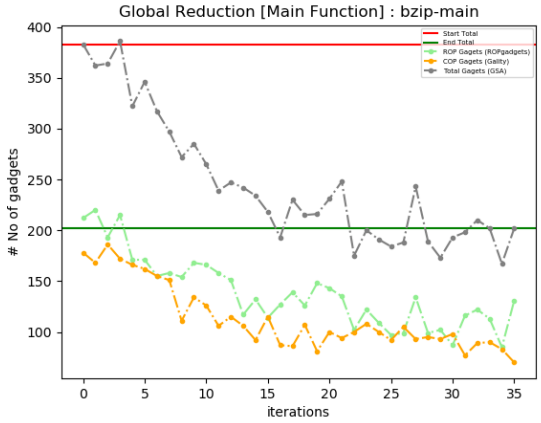

Figure 22: **Chisel** Gadgets Count `main()`

XI. Analysis Metrics

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

A. Static Analysis

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

B. Dynamic Analysis

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

C. Gadgets

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

nettest_bsd							
Libraries/Tools	**Before**	**None**	**Aggressive**	**DeepOCCAM RL Model**	**Non-rec Aggressive**	**Only once**	**IPDSE/IPSCCP Loop Unrolling**
Functions	33	23	24	24	24	24	24
Basic Blocks	979	553	573	573	573	573	573
Instructions Count	6301	3045	3140	3140	3140	3140	3140
Direct Calls	862	402	412	412	412	412	412
External Calls	794	369	377	377	377	377	377
Memory Instructions Load/Store	2859	1350	1396	1396	1396	1396	1396

Table I: Comparison of `DeepOCCAM` with other `OCCAM` Run settings and `OCCAM-T Run (Trimmer)`

httpd							
Libraries/Tools	**Before**	**None**	**Aggressive**	**DeepOCCAM RL Model**	**Non-rec Aggressive**	**Only once**	**IPDSE/IPSCCP Loop Unrolling**
Functions	1083	477	428	430	444	416	441
Basic Blocks	12943	11615	12563	13562	12999	11401	12652
Instructions Count	83238	62428	65842	66521	70773	61667	65252
Direct Calls	22603	5279	5152	5259	5932	5175	5869
External Calls	20712	4152	4563	4628	4787	4116	4625
Memory Instructions Load/Store	17071	16334	18345	17056	18347	16188	17854

Table II: Comparison of `DeepOCCAM` with other `OCCAM` Run settings and `OCCAM-T Run (Trimmer)`

GNU Tree							
Libraries/Tools	**Before**	**None**	**Aggressive**	**DeepOCCAM RL Model**	**Non-rec Aggressive**	**Only once**	**IPDSE/IPSCCP Loop Unrolling**
Functions	52	40	41	44	44	40	42
Basic Blocks	1458	1845	1850	1891	1891	1845	1850
Instructions Count	7442	8957	9152	9286	9286	8957	9365
Direct Calls	1051	1257	1150	1290	1290	1257	1362
External Calls	845	1167	1147	1200	1200	1167	1058
Memory Instructions Load/Store	1944	2362	2344	2344	2344	2222	2452

Table III: Comparison of `DeepOCCAM` with other `OCCAM` Run settings and `OCCAM-T Run (Trimmer)`

XII. Comparision Metrics

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

A. Comparision Metric : Dynamic Analysis

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

B. Comparision Metric : Dynamic Analysis

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

airtun_ng-airtun-ng							
Libraries/Tools	**Before**	**None**	**Aggressive**	**DeepOCCAM RL Model**	**Non-rec Aggressive**	**Only once**	**IPDSE/IPSCCP Loop Unrolling**
Functions	4	4	3	3	3	4	3
Basic Blocks	451	445	450	450	450	445	448
Instructions Count	2521	2481	2523	2523	2523	2481	2523
Direct Calls	328	327	330	330	330	327	330
External Calls	322	321	326	326	326	321	325
Memory Instructions Load/Store	671	658	672	672	672	658	675

Table IV: Comparison of `DeepOCCAM` with other `OCCAM` Run settings and `OCCAM-T Run (Trimmer)`

bzip2							
Libraries/Tools	**Before**	**None**	**Aggressive**	**DeepOCCAM RL Model**	**Non-rec Aggressive**	**Only once**	**IPDSE/IPSCCP Loop Unrolling**
Functions	61	35	54	54	54	35	54
Basic Blocks	2776	2538	3137	3137	3137	2538	3137
Instructions Count	24412	20540	23769	23769	23769	20540	23769
Direct Calls	4714	621	1011	1011	1011	621	1011
External Calls	4575	515	835	835	835	515	835
Memory Instructions Load/Store	5144	5209	5989	5989	5989	5209	5989

Table V: Comparison of `DeepOCCAM` with other `OCCAM` Run settings and `OCCAM-T Run (Trimmer)`

curl							
Libraries/Tools	**Before**	**None**	**Aggressive**	**DeepOCCAM RL Model**	**Non-rec Aggressive**	**Only once**	**IPDSE/IPSCCP Loop Unrolling**
Functions	124	59	62	62	52	59	57
Basic Blocks	2823	2764	4256	4375	3369	2764	3369
Instructions Count	11870	11777	17965	18106	14512	11777	15854
Direct Calls	1786	1696	2423	2500	2005	1696	2145
External Calls	1234	1250	1911	1911	1519	1250	1975
Memory Instructions Load/Store	2503	2511	3698	3858	3048	2511	3625

Table VI: Comparison of `DeepOCCAM` with other `OCCAM` Run settings and `OCCAM-T Run (Trimmer)`

NET UUID Program							
Libraries/Tools	**Before**	**None**	**Aggressive**	**Machine Learning**	**Non-rec Aggressive**	**Only once**	**IPDSE/IPSSCP Loop Unrolling**
Functions	10	9	9	9	9	9	9
Basic Blocks	38	34	34	34	34	34	34
Instructions Count	349	304	304	304	304	304	304
Direct Calls	23	20	20	20	20	20	20
External Calls	12	9	9	9	9	9	9
Memory Instructions Load/Store	141	128	128	128	128	128	128

Table VII: Comparison of `DeepOCCAM` with other `OCCAM` Run settings and `OCCAM-T Run (Trimmer)`

netsh Program Program							
Libraries/Tools	**Before**	**None**	**Aggressive**	**Machine Learning**	**Non-rec Aggressive**	**Only once**	**IPDSE/IPSSCP Loop Unrolling**
Functions	12	11	13	13	13	11	11
Basic Blocks	313	312	332	332	332	312	312
Instructions Count	1319	1315	1417	1417	1417	1315	1315
Direct Calls	195	194	196	196	196	194	194
External Calls	172	171	173	173	173	171	171
Memory Instructions Load/Store	433	431	481	481	481	431	431

Table VIII: Comparison of `DeepOCCAM` with other `OCCAM` Run settings and `OCCAM-T Run (Trimmer)`

Chisel Tool (Final)	**bzip**		**date**		**mkdir**		**rm**		**tree**	
Binary Metrics	**Before**	**After**	**Before**	**After**	**Before**	**After**	**Before**	**After**	**Before**	**After**
ROP Gadgets	646	313	408	166	210	84	485	111	567	405
COP Gadgets	97	55	39	8	7	5	44	5	50	9
JOP Gadgets	6728	1562	5214	877	2282	176	4476	190	2126	775
Total Unique Gadgets (Excluding SYS & Chain)	7374	1930	5626	1046	2492	260	4965	301	2693	1187

Table IX: Dynamic Binary Analysis results for `Chisel` for **Gadgets Count**

Bzip2 Program	Original	Chisel Tool	OCCAM-T (Trimmer)	DeepOCCAM RL Model	OCCAM Aggressive	OCCAM None
ROP Gadgets	646	313	1311	1395	1336	1455
COP Gadgets	97	55	208	226	205	236
JOP Gadgets	6872	1562	3784	3722	3848	4585
Total Unique Gadgets (Excluding SYS & Chain)	7374	1930	5284	5117	5185	6345

Table X: Gadgets count comparision for Bzip2

GNU Tree	Original	Chisel Tool	OCCAM-T (Trimmer)	DeepOCCAM RL Model	OCCAM Aggressive	OCCAM None
ROP Gadgets	567	405	483	567	713	515
COP Gadgets	50	9	19	146	39	83
JOP Gadgets	2126	775	2774	1951	1951	2564
Total Unique Gadgets (Excluding SYS & Chain)	2693	0	3258	2664	2664	3162

Table XI: Gadgets count comparision for GNU Tree

XIII. Observations

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

A. Insights

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

XIV. Conclusion

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques.

Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

A. But finally, Who Won?

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

XV. Other Tools

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

XVI. Project Assets

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram

debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

A. Docker Images

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

B. GitHub Repositories

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

Program debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniquesrogram debloating techniques. Program deblaoting techniques rogram debloating techniques. Program deblaoting techniques

References

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.