Is Software Debloating really effective?

Analysis & Comparison Report

Amit Kumar, 20111012[1], Sumit Lahiri, 19111274[1]

[1]Computer Science & Engineering Dept.
Indian Institute Of Technology, Kanpur

What is Software Debloating?

The Problem

What is Software Debloating?

The Problem

- Multitude of options for a software package with flags [–stats, –debug] etc.

What is Software Debloating?

The Problem

- Multitude of options for a software package with flags [–stats, –debug] etc.
- Do we need all the options and API calls when running for a specific purpose?

What is Software Debloating?

The Problem

- Multitude of options for a software package with flags [–stats, –debug] etc.
- Do we need all the options and API calls when running for a specific purpose?
- What if some options or flags are never used?

What is Software Debloating?

The Problem

- Multitude of options for a software package with flags [–stats, –debug] etc.
- Do we need all the options and API calls when running for a specific purpose?
- What if some options or flags are never used?

Solution

What is Software Debloating?

The Problem

- Multitude of options for a software package with flags [–stats, –debug] etc.
- Do we need all the options and API calls when running for a specific purpose?
- What if some options or flags are never used?

Solution

- To selectively remove such code sections that are not needed for current execution.

What is Software Debloating?

The Problem

- Multitude of options for a software package with flags [–stats, –debug] etc.
- Do we need all the options and API calls when running for a specific purpose?
- What if some options or flags are never used?

Solution

- To selectively remove such code sections that are not needed for current execution.
- Exponentially large number of specific binaries produced, almost all possible combinations!!

How Software Debloating Works?

How Software Debloating Works?

Hinderances

How Software Debloating Works?

Hinderances

- Execution environment modelling.

How Software Debloating Works?

Hinderances

- Execution environment modelling.
- Decide when to remove BBs from the Control Flow Graph ?

How Software Debloating Works?

Hinderances

- Execution environment modelling.
- Decide when to remove BBs from the Control Flow Graph ?
- Can we do better ?

How Software Debloating Works?

Hinderances

- Execution environment modelling.
- Decide when to remove BBs from the Control Flow Graph ?
- Can we do better ?

Rescue

How Software Debloating Works?

Hinderances

- Execution environment modelling.
- Decide when to remove BBs from the Control Flow Graph ?
- Can we do better ?

Rescue

- Debloating tools to rescue !!

How Software Debloating Works?

Hinderances

- Execution environment modelling.
- Decide when to remove BBs from the Control Flow Graph ?
- Can we do better ?

Rescue

- Debloating tools to rescue !!
- Source Code [C or C++] files

How Software Debloating Works?

Hinderances

- Execution environment modelling.
- Decide when to remove BBs from the Control Flow Graph ?
- Can we do better ?

Rescue

- Debloating tools to rescue !!
- Source Code [`C` or `C++`] files
- Specification : What is desired? What all must be executed in the current execution context?

How Software Debloating Works?

Hinderances

- Execution environment modelling.
- Decide when to remove BBs from the Control Flow Graph ?
- Can we do better ?

Rescue

- Debloating tools to rescue !!
- Source Code [`C` or `C++`] files
- Specification : What is desired? What all must be executed in the current execution context?
- Selectively remove those code sections that dont need to be executed as per the specifications provided.

Simple Example

```c
#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;
}
```

After Chisel Tool

Blank Oracle File

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
        int c = 0;

        return 0;
}
```

Before : OCCAM Run

```
Statistics for  before specialization
[CFG analysis]
4 Number of functions
0 Number of specialized functions
0 Number of bounced functions added by devirt
9 Number of basic blocks
52 Number of instructions
4 Number of direct calls
1 Number of external calls
0 Number of assembly calls
0 Number of indirect calls
0 Number of unknown calls
0 Number of loops
0 Number of bounded loops
[Memory analysis]
22 Number of memory instructions
22 Statically safe memory accesses
0 Statically unknown memory accesses
```

After : OCCAM Run

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

Our Project

Our Project

Objectives

Our Project

Objectives

- Understanding of debloating techniques.

Our Project

Objectives

- Understanding of debloating techniques.
- Run state-of-the-art debloating tools against benchmarks for comparison between different approaches and techniques

Our Project

Objectives

- Understanding of debloating techniques.
- Run state-of-the-art debloating tools against benchmarks for comparison between different approaches and techniques
- Implementation of `DeepOCCAM` from `OCCAM` based on `DeepOCCAM` Paper.

Tools

Our Project

Objectives

- Understanding of debloating techniques.
- Run state-of-the-art debloating tools against benchmarks for comparison between different approaches and techniques
- Implementation of `DeepOCCAM` from `OCCAM` based on `DeepOCCAM` Paper.

Tools

- `OCCAM` : Automated Software Winnowing

Our Project

Objectives

- Understanding of debloating techniques.
- Run state-of-the-art debloating tools against benchmarks for comparison between different approaches and techniques
- Implementation of `DeepOCCAM` from `OCCAM` based on `DeepOCCAM` Paper.

Tools

- `OCCAM` : Automated Software Winnowing
- `Trimmer` : Input Specialization, Specialized Loop Unrolling, Constant Propagation.

Our Project

Objectives

- Understanding of debloating techniques.
- Run state-of-the-art debloating tools against benchmarks for comparison between different approaches and techniques
- Implementation of `DeepOCCAM` from `OCCAM` based on `DeepOCCAM` Paper.

Tools

- `OCCAM` : Automated Software Winnowing
- `Trimmer` : Input Specialization, Specialized Loop Unrolling, Constant Propagation.
- `Chisel` : Reinforcement Learning based Delta Debugging on a set of Tests.

Our Project

Objectives

- Understanding of debloating techniques.
- Run state-of-the-art debloating tools against benchmarks for comparison between different approaches and techniques
- Implementation of `DeepOCCAM` from `OCCAM` based on `DeepOCCAM` Paper.

Tools

- `OCCAM` : Automated Software Winnowing
- `Trimmer` : Input Specialization, Specialized Loop Unrolling, Constant Propagation.
- `Chisel` : Reinforcement Learning based Delta Debugging on a set of Tests.
- `DeepOCCAM` : An extension to `OCCAM`, based on Reinforcement Learning based Specialization Action.

Implementation

Implementation

Pipelines

Implementation

Pipelines

- OCCAM

Implementation

Pipelines

- `OCCAM`
- `OCCAM-T` : A modified run of OCCAM with `--unroll-loop`, `--ipdce`, `specialize=true`

Implementation

Pipelines

- `OCCAM`
- `OCCAM-T` : A modified run of OCCAM with `--unroll-loop`, `--ipdce`, `specialize=true`
- `Chisel` : Modified to dump chunks and other metrics data

Implementation

Pipelines

- `OCCAM`
- `OCCAM-T` : A modified run of OCCAM with `--unroll-loop`, `--ipdce`, `specialize=true`
- `Chisel` : Modified to dump chunks and other metrics data
- `DeepOCCAM` : Developed from base `OCCAM` tool.

Diagrams

Chisel

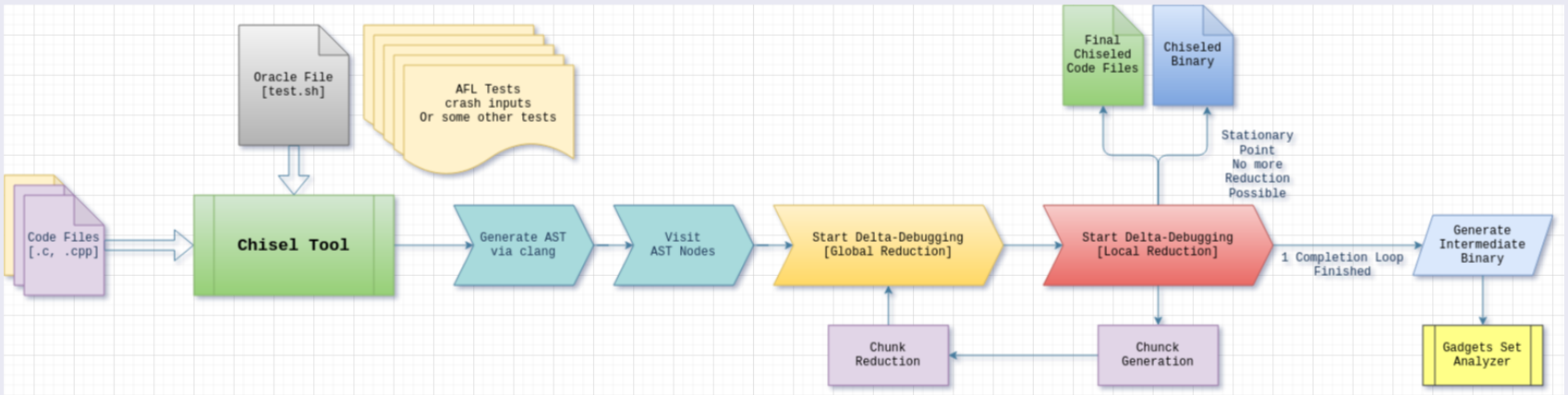


Figure: **Chisel Pipeline** : Setup `Chisel` runs on `Chisel-Benchmarks` & other examples

Diagrams

OCCAM

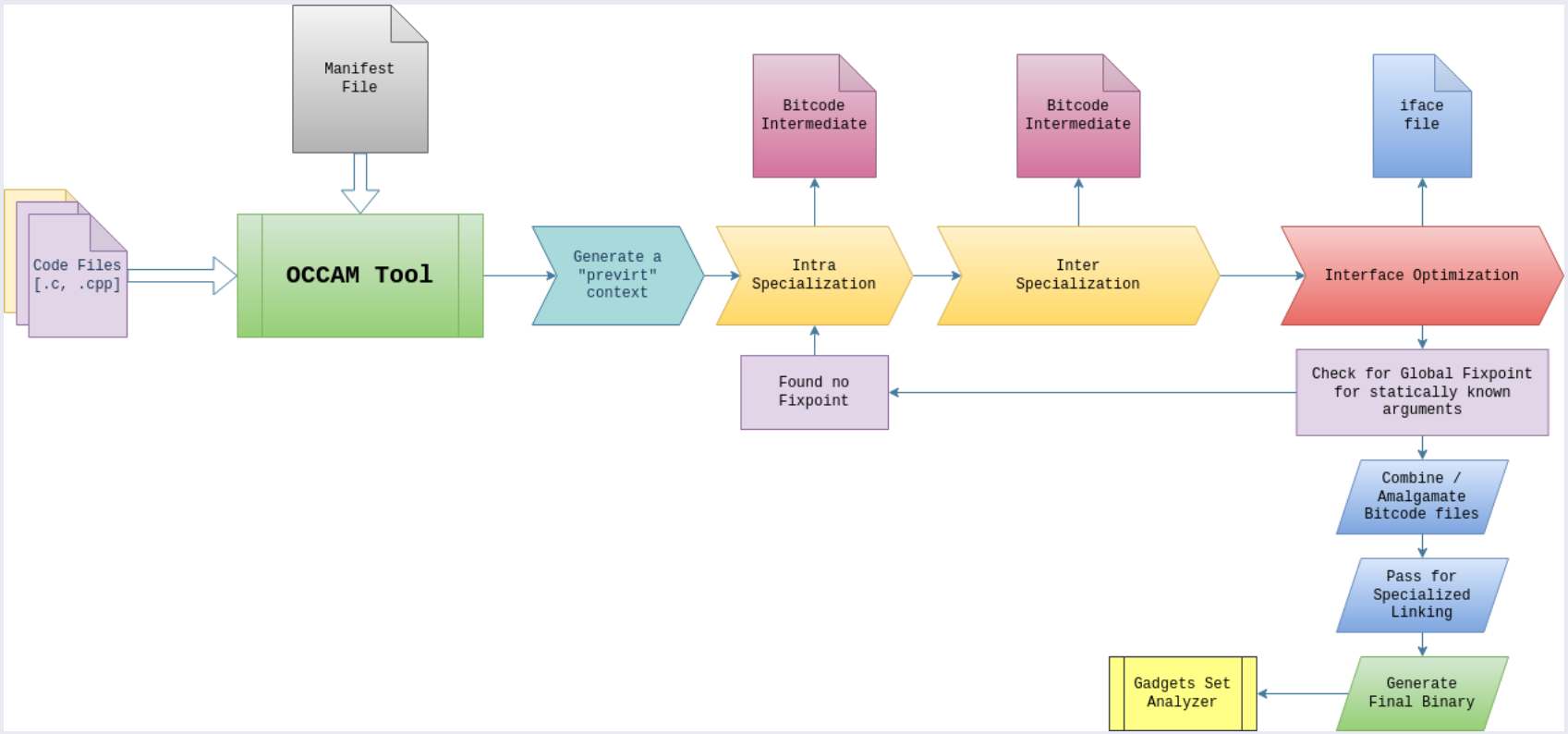


Figure: **OCCAM Pipeline** : Setup `OCCAM` runs on `OCCAM-Benchmarks` & other examples

Diagrams

Trimmer

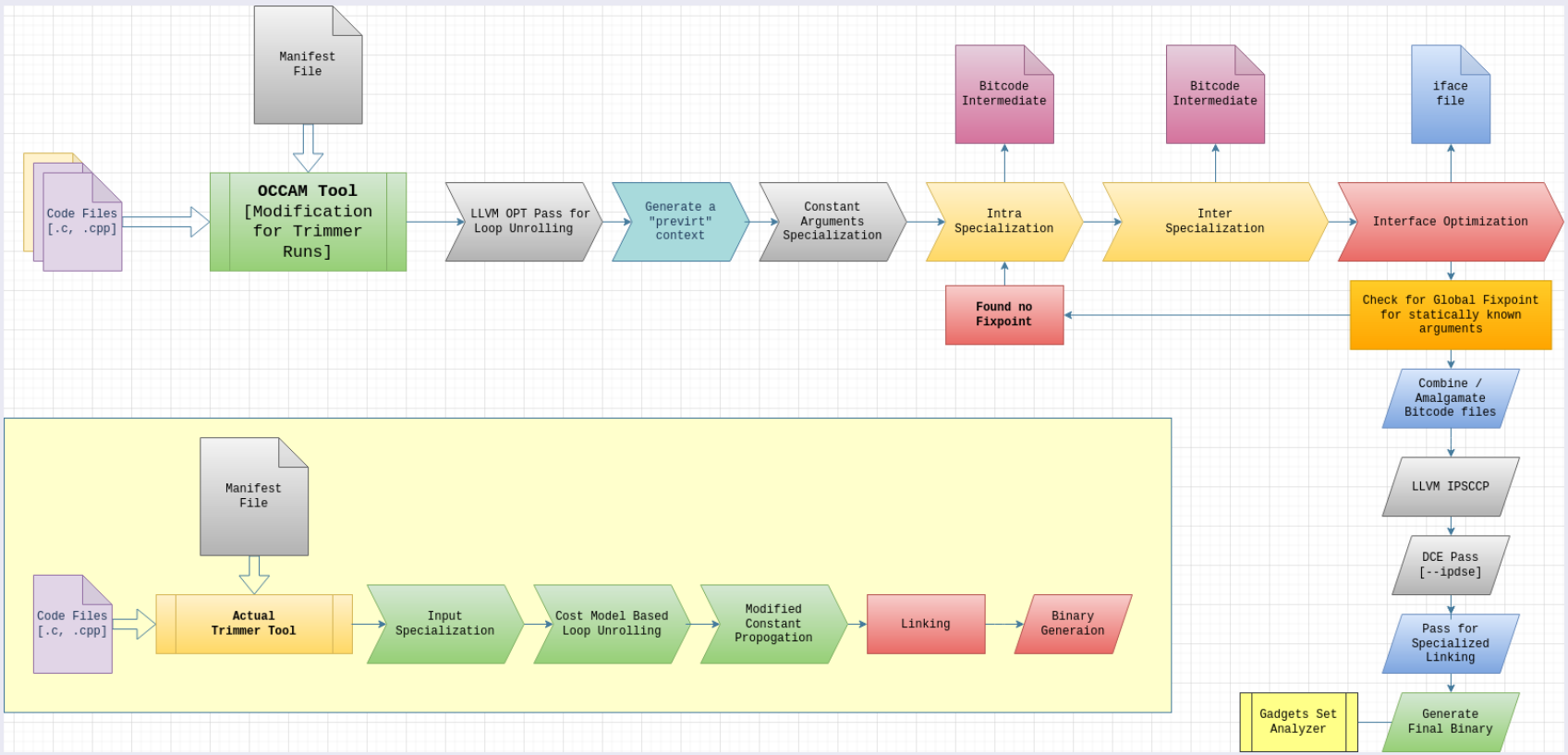
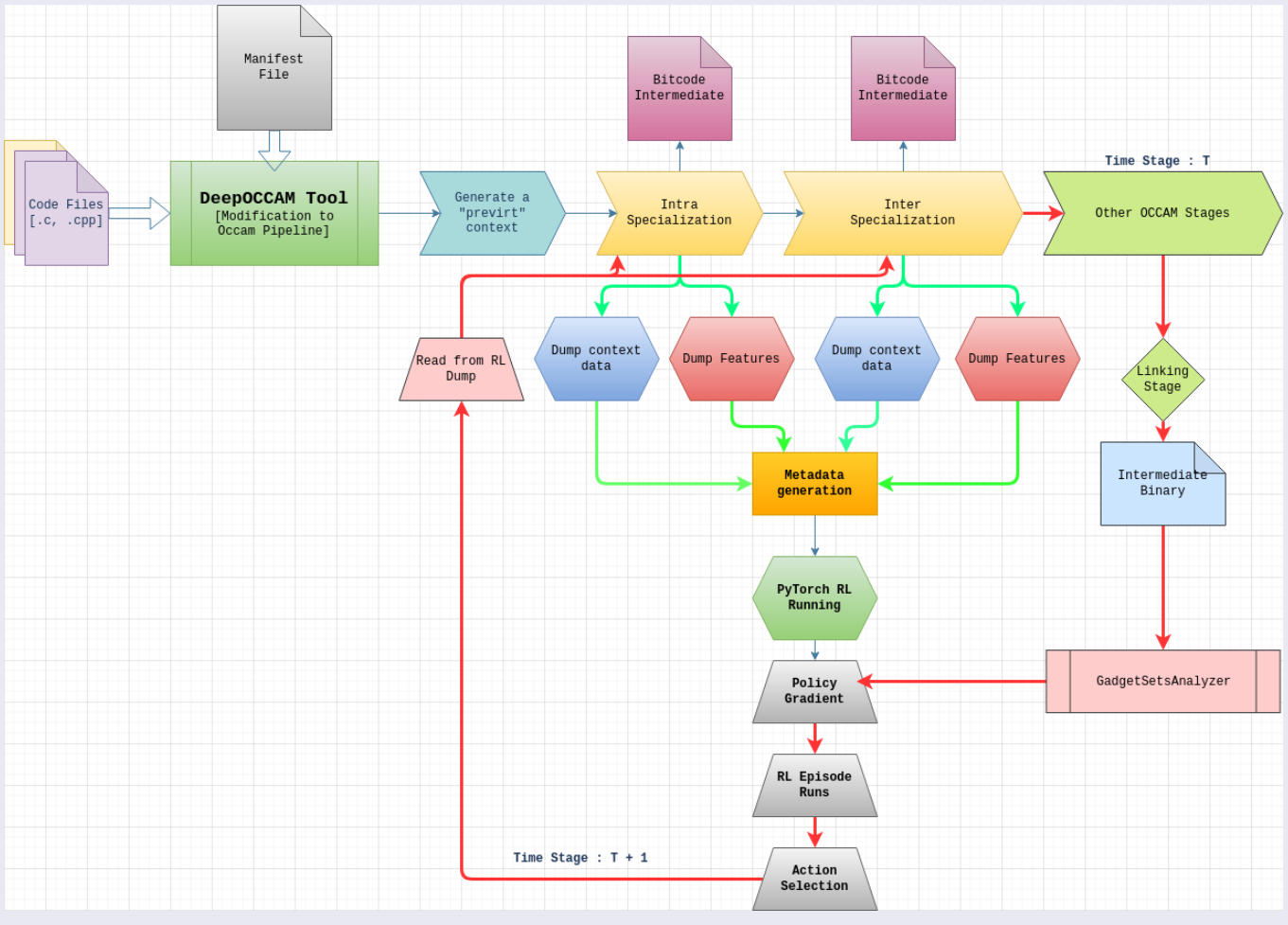


Figure: **Trimmer Runs** : Setup `Trimmer` runs on `OCCAM-Benchmarks` only

Diagrams

DeepOCCAM



Static Analysis

httpd program

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)`

Static Analysis

curl program

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)`

Dynamic Analysis

Chisel Runs

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**

Dynamic Analysis

Full Comparison

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: Dynamic Binary Analysis : 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	1189	3258	2664	2664	3162

Table XI: Dynamic Binary Analysis : Gadgets Count comparision for GNU Tree

Chisel Learning

Plots

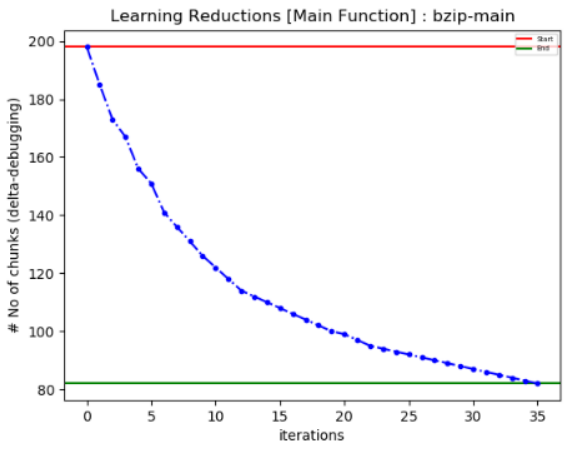


Figure 20: **Chisel** Learning bzip `main()`

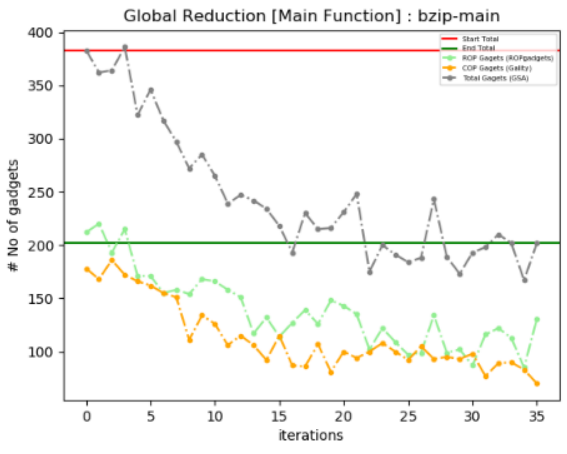


Figure 23: **Chisel** Gadgets Count `main()`

Conclusion

Insights & Closing Remarks

Conclusion

Insights & Closing Remarks

- Did Chisel Won?

Conclusion

Insights & Closing Remarks

- Did Chisel Won? No !!,

Conclusion

Insights & Closing Remarks

- Did Chisel Won? No !!, 14 hrs to run `bzip2`,

Conclusion

Insights & Closing Remarks

- Did Chisel Won? No !!, 14 hrs to run `bzip2`, 1 day to run (uniq)

Conclusion

Insights & Closing Remarks

- Did Chisel Won? No !!, 14 hrs to run `bzip2`, 1 day to run (uniq)
- How to decide which tool to use based on the comparison we showed?

Conclusion

Insights & Closing Remarks

- Did Chisel Won? No !!, 14 hrs to run `bzip2`, 1 day to run (uniq)
- How to decide which tool to use based on the comparison we showed?
- Depends on our objective.

Conclusion

Insights & Closing Remarks

- Did Chisel Won? No !!, 14 hrs to run `bzip2`, 1 day to run (uniq)
- How to decide which tool to use based on the comparison we showed?
- Depends on our objective.
- Recommend `Chisel` : With AFL Fuzzing or EGT [Symbolic Execution]

Conclusion

Insights & Closing Remarks

- Did Chisel Won? No !!, 14 hrs to run `bzip2`, 1 day to run (uniq)
- How to decide which tool to use based on the comparison we showed?
- Depends on our objective.
- Recommend `Chisel` : With AFL Fuzzing or EGT [Symbolic Execution]
- Recommend `OCCAM Variants` : Writing manifest simple!, less effective on gadgets reduction.