Lightweight, Multi-Stage, Compiler-Assisted Application Specialization (LMCAS)

Mohannad Alhanahnah and Somesh Jha

University of Wisconsin-Madison
Demands for Tiny & Specialized Utilities

BusyBox

Cloud with dollar sign

Use Case
LMCAS
Q&A
demo
Tiny Utilities Generated Manually
Our Goal
Compilers are Powerful
Disciplined Software Development

Meinicke et al., Exploring differences and commonalities between feature flags and configuration options. ICSE-SEIP'20
LMCAS Pillars

Program

- Configuration
- Neck – end of config
- Main Logic

LMCAS

Specialized Utilities

GCC

https://github.com/Mohannadcse/LMCAS_Demo
Agenda

• Use Case
• Introducing LMCAS
• Questions
• Demo
Survey

• Do you follow Disciplined Software Development?

YES □

NO □
Agenda

• Use Case
  • Introducing LMCAS
  • Questions
• Demo
Use Case (Network Monitoring)
Use Case (Network Monitoring)

Specialization = High Efficiency
Other Use Cases
Agenda

• Use Case
• Introducing LMCAS
• Questions
• Demo
LMCAS Workflow (Partial Evaluation)

- Supplied Inputs
- LMCAS
- Specialized Program
- Delayed Inputs
LMCAS Workflow - Examples

```
tcpdump
-i eth0
-c 5

WC
-l
-c

Specialized tcpdump

Specialized wc

<fileName>

Intuition

Use Case

LMCAS

Q&A

demo
Illustrative Example

- scaled-down version of the `wc` utility
  1. Line count
  2. Char count

```c
struct Flags {
    char count_chars;
    int count_lines;
};
int total_lines = 0;
int total_chars = 0;
int main(int argc, char** argv){
    struct Flags *flag;
    flag = malloc(sizeof(struct Flags));
    flag->count_chars = 0;
    flag->count_lines = 0;
    if (argc >= 2){
        for (int i = 1; i < argc; i++) {
            if (!strcmp(argv[i], "-c")) flag->count_chars = 1;
            if (!strcmp(argv[i], "-l")) flag->count_lines = 1;
        }
    }
    char buffer[1024];
    while (fgets(buffer, 1024, stdin)){
        if (flag->count_chars) total_chars += decodeChar(buffer);
        if (flag->count_lines) total_lines++;
    }
    if (flag->count_chars) printf("#Chars= %d", total_chars);
    if (flag->count_lines) printf("#Lines= %d", total_lines);
    return 0;
}
```
LMCAS Approach

```c
struct Flags {
    char count_chars;
    int count_lines;
}
int total_lines = 0;
int total_chars = 0;
int main(int argc, char** argv)
{
    struct Flags *flag;
    flag = malloc(sizeof(struct Flags));
    flag->count_chars = 0;
    flag->count_lines = 0;
    if (argc >= 2){
        for (int i = 1; i < argc; i++) {
            if (!strcmp(argv[i], "-c")) flag->count_chars = 1;
            if (!strcmp(argv[i], "-l")) flag->count_lines = 1;
        }
    }
    char buffer[1024];
    while (fgets(buffer, 1024, stdin)){
        if (flag->count_chars) total_chars += decodeChar(buffer);
        if (flag->count_lines) total_lines++;
        if (flag->count_chars) printf("#Chars = %d", total_chars);
        if (flag->count_lines) printf("#Lines = %d", total_lines);
    }
    return 0;
}
```

**Configuration Logic**

```
struct Flags {
    char count_chars;
    int count_lines;
}
int total_lines = 0;
int total_chars = 0;
int main(int argc, char** argv){
    struct Flags *flag;
    flag = malloc(sizeof(struct Flags));
    flag->count_chars = 0;
    flag->count_lines = 0;
    if (argc >= 2){
        for (int i = 1; i < argc; i++) {
            if (!strcmp(argv[i], "-c")) flag->count_chars = 1;
            if (!strcmp(argv[i], "-l")) flag->count_lines = 1;
        }
    }
}
```

**Interpretation**

1. Neck
2. Interpretation
3. Enforcing program state

**Main Logic**

```c
char buffer[1024];
while (fgets(buffer, 1024, stdin)){
    if (count_chars) total_chars += decodeChar(buffer);
    if (count_lines) total_lines++;
    if (count_chars) printf("#Chars = %d", total_chars);
    if (count_lines) printf("#Lines = %d", total_lines);
    return 0;
```
LMCAS Architecture

Modified KLEE

Neck Miner

Neck Identification (currently manual)

LLVM IR

Partial Interpretation

Partial Evaluation

Constant Conversion

Multi-Stage Simplification

Specialized Program

LLVM Passes

| Variable            | Type | Scope   | Value |
|---------------------|------|---------|-------|
| total_lines         | int  | Global  | 0     |
| total_chars         | int  | Global  | 0     |
| flag->count_lines   | int  | Local   | 1     |
| flag->count_chars   | char | Local   | 0     |
Neck Properties

• Neck should be an articulation point in the CFG:
  • Dominator of main logic nodes
  • Always executed
    • Reachable from the entry
  • Executed once
    • Outside any loop structure

```c
struct Flags {
  char count_chars;
  int count_lines;
};
int total_lines = 0;
int total_chars = 0;
int main(int argc, char** argv){
  struct Flags *flag;
  flag = malloc(sizeof(struct Flags));
  flag->count_chars = 0;
  flag->count_lines = 0;
  if (argc >= 2){
    for (int i = 1; i < argc; i++) {
      if (!strcmp(argv[i], "-c")) flag->count_chars = 1;
      if (!strcmp(argv[i], "-l")) flag->count_lines = 1;
    }
  }
  char buffer[1024];
  while (fgets(buffer, 1024, stdin)){
    if (count_chars) total_chars += sizeof(buffer);
    if (count_lines) total_lines++;
  }
  if (count_chars) printf("# chars = %d\n", total_chars);
  if (count_lines) printf("# Lines = %d\n", total_lines);
```
Partial Interpreter

Modified KLEE
Program Partial State

Configuration Logic

```c
struct Flags {
    char count_chars;
    int count_lines;
    int total_lines = 0;
    int total_chars = 0;
    int main(int argc, char* argv){
        struct Flags *flag;
        flag = malloc(sizeof(struct Flags));
        flag->count_chars = 0;
        flag->count_lines = 0;
        if (argc >= 2){
            for (int i = 1; i < argc; i++) { 
                if (strcmp(argv[i], "-c")) flag->count_chars = 1;
                if (strcmp(argv[i], "-l")) flag->count_lines = 1;
            }
        }
    }
}
```

Global Variables

- `char buffer[1024];`
- `while (fgets(buffer, 1024, stdin)){`  
  - `(count_chars) += sizeOf(buffer);`
  - `(count_lines) += total_lines++;`
- `if (count_chars) printf("%d\n", total_chars);`
- `if (count_lines) printf("%d\n", total_lines);`

Stack Variables

- `char buffer[1024];`
- `while (fgets(buffer, 1024, stdin)){`  
  - `(count_chars) += sizeOf(buffer);`
  - `(count_lines) += total_lines++;`
- `if (count_chars) printf("%d\n", total_chars);`
- `if (count_lines) printf("%d\n", total_lines);`

Stored as txt files:

-globals=gbls.txt
-primitive stack=primitiveLocals.txt
-struct stack=customizedLocals.txt
-ptr to struct=ptrToStructLocals.txt
-ptr to primitive=ptrToPrimitiveLocals.txt
-string Vars=stringVars.txt
-nested structs=nestedStructLocals.txt

Intuition

Use Case

LMCAS

Q&A

demo
Partial Interpreter Implementation

KLEE 2.1

LLVM 10.0
Compiler-Assisted Optimizations

Constant Conversion
Multi-stage Simplifications
Constant Conversion (CC)

### Configuration Logic

```c
struct Flags {
    char count_chars;
    int count_lines;
};
int total_lines = 0;
int total_chars = 0;
int main(int argc, char** argv){
    struct Flags *flag;
    flag = malloc(sizeof(struct Flags));
    flag->count_chars = 0;
    flag->count_lines = 0;
    if (argc >= 2){
        for (int i = 1; i < argc; i++) {
            if (strstr(argv[i], "-c")) flag->count_chars = 1;
            if (strstr(argv[i], "-l")) flag->count_lines = 1;
        }
    }
    char buffer[1024];
    while (fgets(buffer, 1024, stdin)){
        if (count_chars) total_chars += sizeof(buffer);
        if (count_lines) total_lines++;
    }
    if (count_chars) printf("# chars = %d\n", total_chars);
    if (count_lines) printf("# Lines = %d\n", total_lines);
    return 0;
}
```

### Backward CC

- **-globals=gbls.txt**
- **-plocals=primitiveLocals.txt**
- **-clocals=customizedLocals.txt**
- **-ptrStructlocals=ptrToStructLocals.txt**
- **-ptrToPrimLocals=ptrToPrimitiveLocals.txt**
- **-stringVars=stringVars.txt**
- **-nestedStrcts=nestedStructLocals.txt**

### Forward CC

| Variable     | Type  | Scope | Value |
|--------------|-------|-------|-------|
| total_lines  | int   | Global| 0     |
| total_chars  | int   | Global| 0     |
| flag->count_lines | int | Local  | 1     |
| flag->count_chars   | char |       | 0     |
CC Pre-neck

```c
int main (int argc, char **argv)
{
    struct rm_options x;
    rm_option_init (&x);
    while ((c = getopt_long (argc, argv, "dfirv\", long_opts, NULL)) != -1)
    {
        switch (c)
        {
        case 'd':
            x.remove_empty_directories = true;
            break;
        case 'f':
            x.interactive = RMI_NEVER;
            x.ignore_missing_files = true;
            prompt_once = false;
            break;
        case 'i':
            x.interactive = RMI_ALWAYS;
            x.ignore_missing_files = false;
            prompt_once = false;
            break;
        ...

    static void rm_option_init (struct rm_options *x)
    {
        x->ignore_missing_files = false;
        x->interactive = RMI_SOMETIMES;
        x->one_file_system = false;
        x->remove_empty_directories = false;
        x->recursive = false;
        x->root_dev_ino = NULL;
        x->preserve_all_root = false;
        x->stdin_tty = isatty (STDIN_FILENO);
        x->verbose = false;
        x->require_restore_cwd = false;
    }
```
Multi-stage Simplifications

- **Constant Propagation**
  - Standard LLVM pass

- **Simplifying CFG**
  - Standard LLVM Pass

- **Cleaning up**
  - Customized LLVM Pass

Intuition

Use Case

LMCAS

Q&A

demo
Cleaning up

Removing unused functions

- Iterate the Call Graph
- Check the number of users of each function
- Remove the function if number of users is zero
  - function pointer won’t be removed

Removing unused variables in the remained functions

- Local and global

We used users() LLVM API
LMCAS Power

- Lightweight approach
  - No test cases/heuristics are required

- Reduce programs size
  - Binary
  - Functions
  - Basic blocks
  - Instructions

- Reduce attack surface
  - Eliminate known CVEs
  - Reduce gadgets

- Correct functionality
  - Except the neck
  - Ongoing work

- Fully automated

- Supports Command-line programs
Agenda

• Use Case
• Introducing LMCAS
• Questions
• Demo
LMCAS - Summary

Supplied Inputs → LMCAS → Specialized Program

Delayed Inputs → LMCAS

LLVM IR → C

Neck Miner + LLVM IR → LLVM IR

Partial Interpretation → Partial Evaluation → Constant Conversion → Multi-Stage Simplification

Partial State → Supplied Inputs

Specialized Program

Delayed Inputs
Agenda

• Use Case
• Introducing LMCAS
• Questions
• Demo
Demo Roadmap

- Two debloating scenarios
- See size reduction statistics
- See gadgets reduction statistics
- Verify the functionality of the debloated program
Debloating Scenarios

Demo Scenarios

Illustrative Example “wc”
- Count # lines and chars

Networking “tcpdump”
- Monitor loopback interface
- Capture only 5 packets

LMCAS Docker and programs used in the demo are available at:
https://github.com/Mohannadcse/LMCAS_Demo

18 Programs
Specializing GNU wc
Specializing tcpdump
Summary

LMCAS specializes programs
- Compiler Optimizations
- Disciplined Implementation

LMCAS Benefits
- Lightweight approach
- Reducing attack surface
- Reducing program size

Ongoing project
- Automating the neck identification
- Leveraging data flow analysis
- Covering more real-world use cases
More Info

GitHub: https://github.com/Mohannadcse/LMCAS_Demo

arXiv: https://arxiv.org/abs/2109.02775

Email: mohannad@cs.wisc.edu

Twitter: Malhanahnah