SymDefFix - Sound Automatic Repair Using Symbolic Execution

Tareq Mohammed Nazir and Martin Pinzger

Software Engineering Research Group, University of Klagenfurt, Austria

3rd International KLEE Workshop on Symbolic Execution

Automatic repair of programs

- Developers spend ~90% of their time to manually understand and fix bugs

- Several approaches for automatically repairing programs, e.g., GenProg, SemFix, DirectFix, and ExtractFix

Drawbacks of existing approaches

- Many approaches, e.g., GenProg, SemFix, DirectFix, suffer from overfitting - they generate patches that pass the test suite

- Constraints based automatic repair, e.g. ExtractFix, addresses overfitting, but need a test case to trigger the bug and output the constraint(s) for generating the patch(es)

- Quality of the generated patch(es) is still questionable

Example of a patch generated by ExtractFix

Program:

```
size_t LOWFAT_GLOBAL_MS__heap_overflow__malloc_7;
char* _malloc(int size){
  char* buf = (char*)malloc((
    {LOWFAT_GLOBAL_MS__heap_overflow__malloc_7 = size;
     LOWFAT_GLOBAL_MS__heap_overflow__malloc_7;}
  ));
  return buf;
}

int main(int argc, char *argv[]){
  char *buffer = _malloc(5);
  // ......
  char* content = argv[1];
  int content_size = strlen(content);
  for (i; i<content_size; i++)
    buffer[i] = content[i];
  // ......
}
```

Input :
HelloWorld!

Crash Location & Generated Constraints:
i < LOWFAT_GLOBAL_MS__heap_overflow__malloc_7

Generated Patch:

```
- for (i; i<sizeof(content); i++)
---
+ for (i;(((i)<sizeof(content)) &&((i)<
(LOWFAT_GLOBAL_MS__heap_overflow__malloc_7)));i++)
```

Better Patch could be:

```
- char *buffer = _malloc(5);
  // ......
    char* content = argv[1];
    int content_size = strlen(content);
+ char *buffer = _malloc(content_size);
  for (i; i<content_size; i++)
    buffer[i] = content[i];
  // ......
```

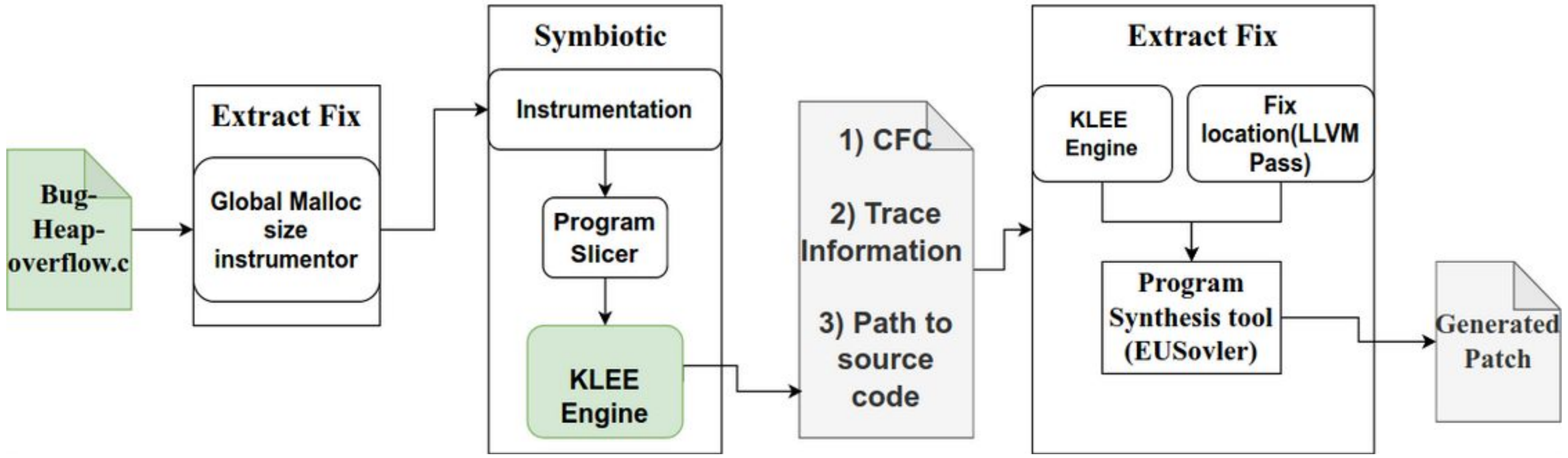
Goal of SymDefFix

- Create an automatic program repair tool that
  - detects bugs in the source code using static analysis
  - generates high quality patches
  - does not overfit a test suite

Research hypothesis of SymDefFix

- We can use symbolic execution to get more information about the bug
  - to determine the crash free constraint(s) and fix location(s)
  - to generate more accurate patches

SymDefFix approach



- Questions addressed in this work:
  - RQ1: How can we derive crash free constraints using KLEE symbolic execution engine?
  - RQ2: How can we derive the interprocedural calls using KLEE symbolic execution engine?

RQ1: Output the CFC using KLEE

Implemented it using the dump stack feature of KLEE

Example:

```
size_t GLOBAL_MS__heap_overflow__malloc_7;
char* _malloc(int size){
  char* buf = (char*)malloc((
    {GLOBAL_MS__heap_overflow__malloc_7 = size;
     GLOBAL_MS__heap_overflow__malloc_7;}
  ));
  return buf;
}

int main(int argc, char *argv[]){
  char *buffer = _malloc(5);
  // ......
  char* content[10];
  int content_size = strlen(content);
  for (i; i<content_size; i++)
    buffer[i] = content[i];
  // ......
}
```

Format of constraints:
filename.c:function_name:crash_line_number#constraints

CFC output from KLEE engine:

cfc.out information: heap_overflow.c:main:30#(i < GLOBAL_MS__heap_overflow__malloc_7)

RQ2: Output the inter-procedure calls using KLEE

Implemented it by utilizing the Executor class of KLEE

Example:

```
size_t GLOBAL_MS__heap_overflow__malloc_7;
char* _malloc(int size){
  char* buf = (char*)malloc((
    {GLOBAL_MS__heap_overflow__malloc_7 = size;
     GLOBAL_MS__heap_overflow__malloc_7;}
  ));
  return buf;
}

int main(int argc, char *argv[]){
  char *buffer = _malloc(5);
  // ......
  char* content[10];
  int content_size = strlen(content);
  for (i; i<content_size; i++)
    buffer[i] = content[i];
  // ......
}
```

Calls:

```
IN >>>> main :
: Success
IN >>>> _malloc
: Success
OUT >>>> _malloc
:Success
OUT >>>> main
: Success
```

Generated patch(es)

```
- for (i; i<sizeof(content); i++)
---
+ for (i;(((i)<sizeof(content)) &&((i)< (GLOBAL_MS__heap_overflow__malloc_7)));i++)
```

Conclusions

- Used KLEE symbolic execution to get more information about the bug to compute the crash free constraints and inter-procedural call trace
- Replaced the dynamic analysis part of ExtractFix with a static analysis approach
- SymDefFix obtained the same patch as output by ExtractFix
- Future Work
  - Consider all symbolically executed (error) paths
  - Improve the algorithm(s) to determine the fix locations
  - Improve the algorithm(s) to synthesize (generate) the patches
  - Consider other types of bugs, e.g., divide by zero, null pointer, pointer dereferencing issues etc.