Reproducer Add-On for Focaccia

Automated Test Case Generation for Emulators Using Symbolic Execution

Alp Berkman
Supervisor: Prof. Dr.-Ing. Pramod Bhatotia
Advisor: Sebastian Reimers, M.Sc. & Theofilos Augoustis, M.Sc.
Chair of Computer Systems
https://dse.in.tum.de/

15.11.2023 – 15.03.2024

Motivation

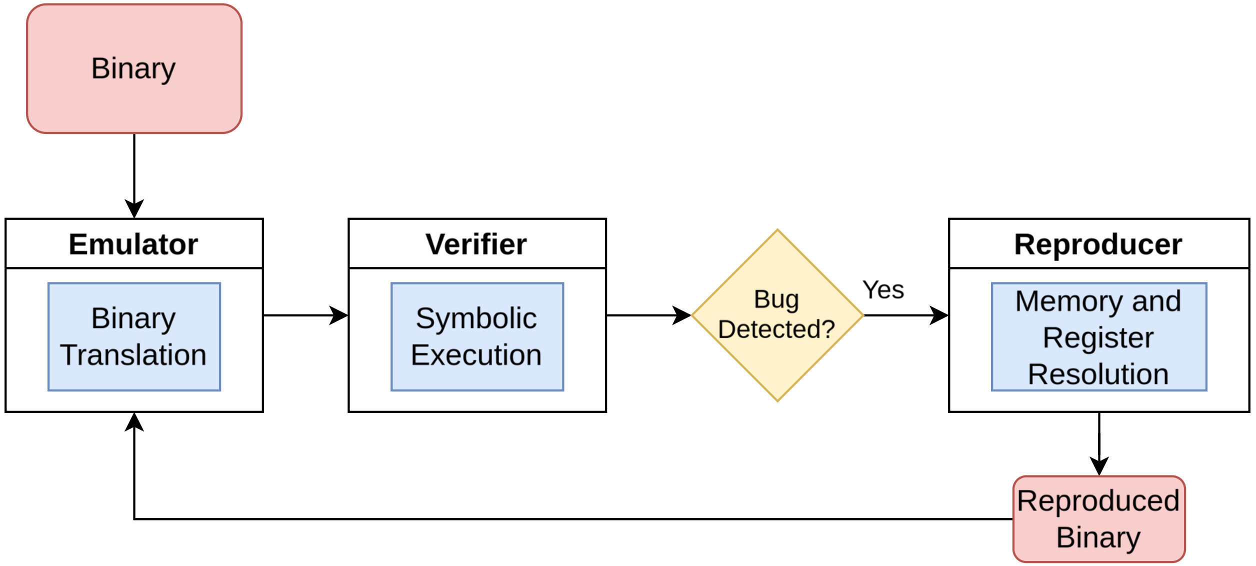
- New architectures like ARM and RISC-V
- Emulation of x86 ISA on other ISAs using virtual machines
- Bugs occur during binary translation
- Bugs in virtual machines are hard to debug, due to their nature

How to Reliably and Automatically Test Emulators

- Unit tests:
  - Written as a reaction to bugs
  - Requires manual work
  - Likely to have it’s own bugs
- Fuzzing:
  - Takes very long
  - Not guaranteed to work
- Not systematic

Problem statement

How to automatically isolate bugs that occur during translation



Outline

- ~~Motivation~~
- Background
  - Focaccia Verifier
  - Binary Translation
  - Symbolic Execution
- Design
- Implementation
- Evaluation

Background: Focaccia Verifier

- Developed by TUM’s Systems Research Group
- Checks the correctness of emulators
- Uses concolic (concrete+symbolic) execution

Background: Symbolic Execution

- A method to analyze the behavior of a program
- Uses symbolic values, represents all possible input values
- Creates a tree of execution paths
-
- Tests programs systematically
- Very computationally intensive
- Path explosion for bigger programs

Background: Binary Translation

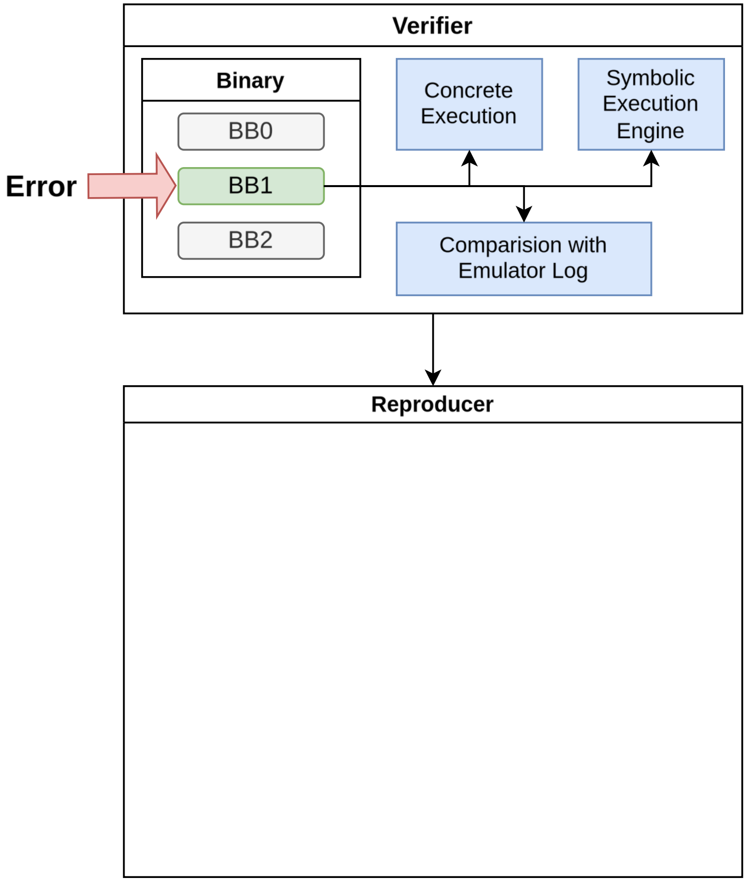
- Enables running a binary compiled for a different ISA
- Translates instructions from guest ISA to host ISA
- Well known examples:
  - QEMU
  - Rosetta

Outline

- ~~Motivation~~
- ~~Background~~
- Design
  - System design
- Implementation
- Evaluation

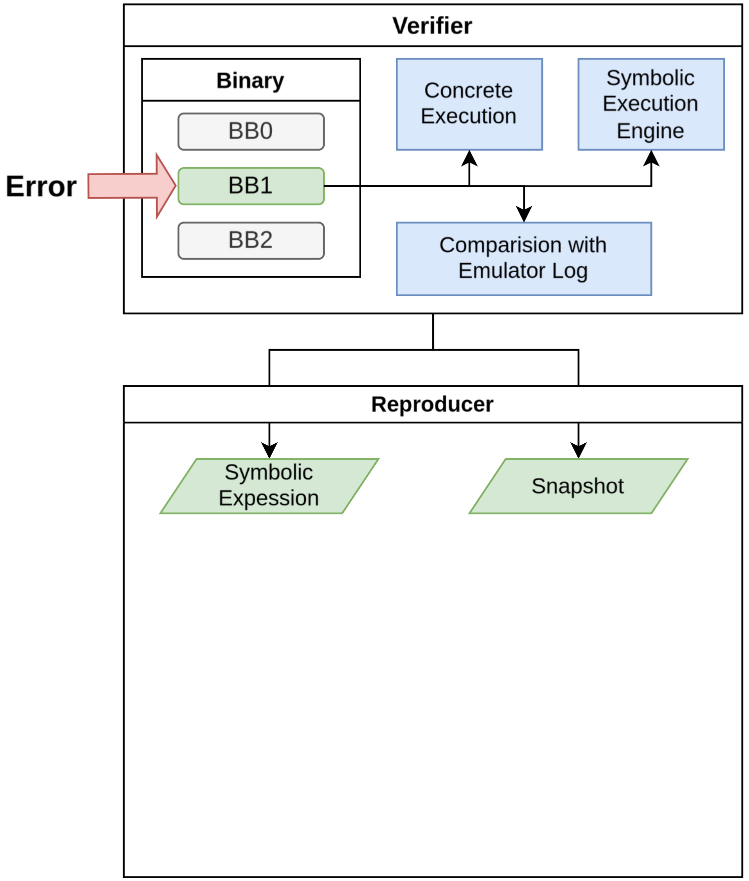
System Design

- Add-on to Focaccia verifier



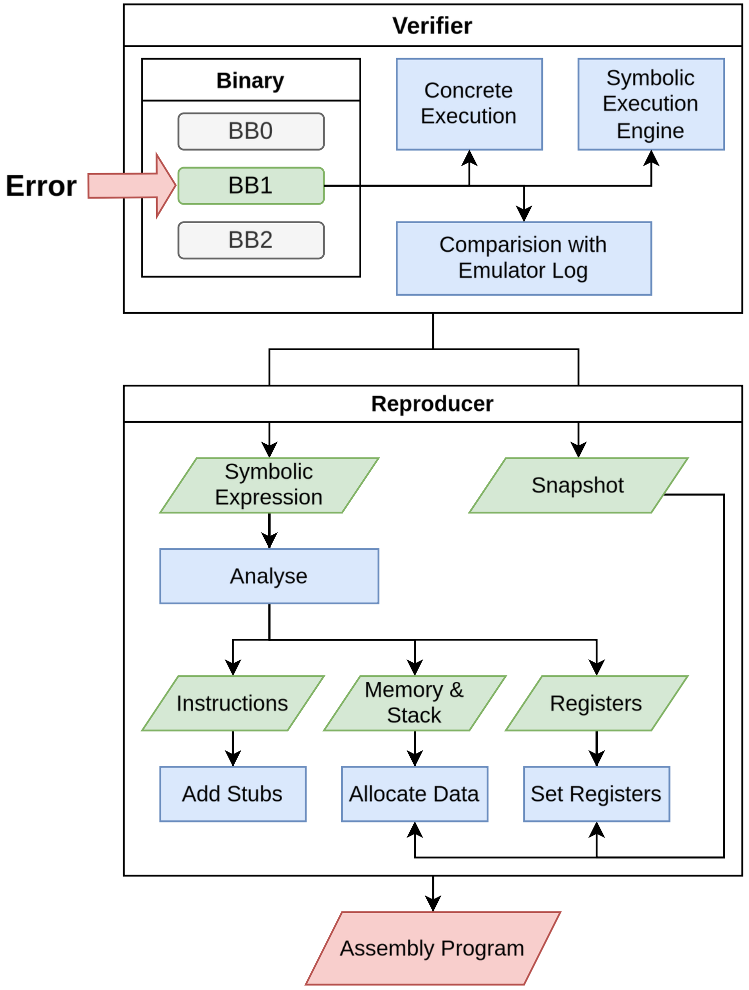
System Design

- Add-on to Focaccia verifier
- Receives:
  - Symbolic expression
  - Snapshot



System Design

- Add-on to Focaccia verifier
- Receives:
  - Symbolic expression
  - Snapshot
- Produces assembly instructions:
  - Create a similar environment
  - Trigger the bug



Outline

- ~~Motivation~~
- ~~Background~~
- ~~Design~~
- Implementation
- Evaluation

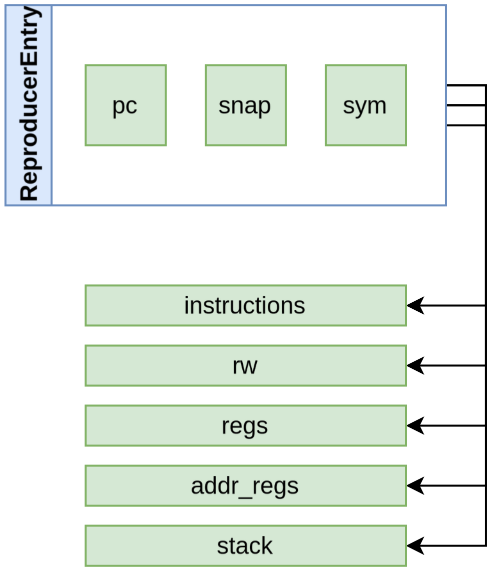
Implementation

- Written in Python
- Made out of two parts

Implementation

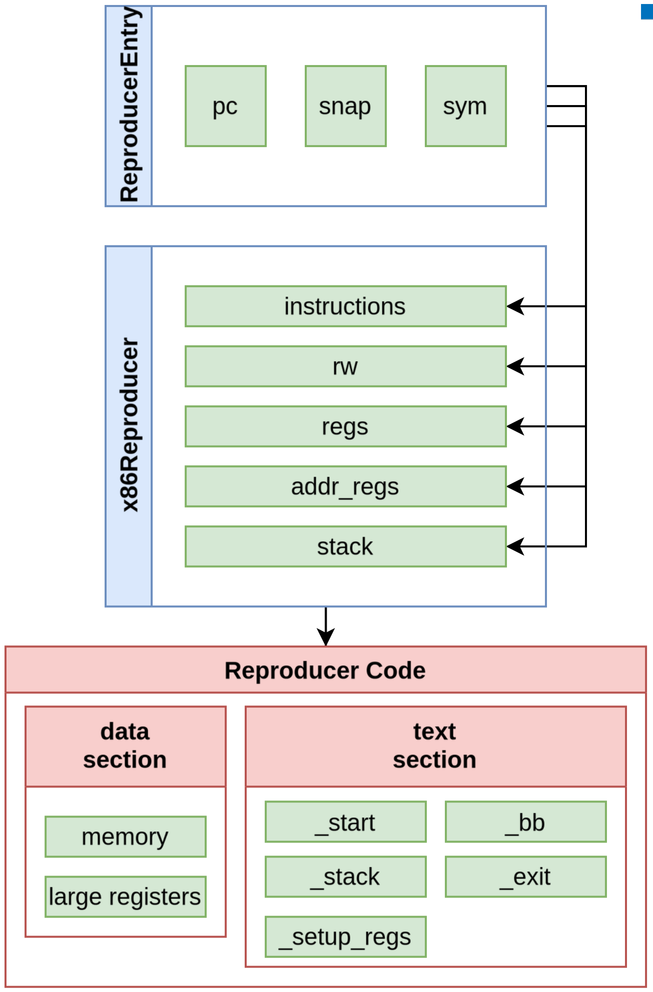
- Written in Python

- Made out of two parts
    - Generic part for extracting values



Implementation

- Written in Python
- Made out of two parts
  - Generic part for extracting values
  - Hardware specific part for creating assembly instructions



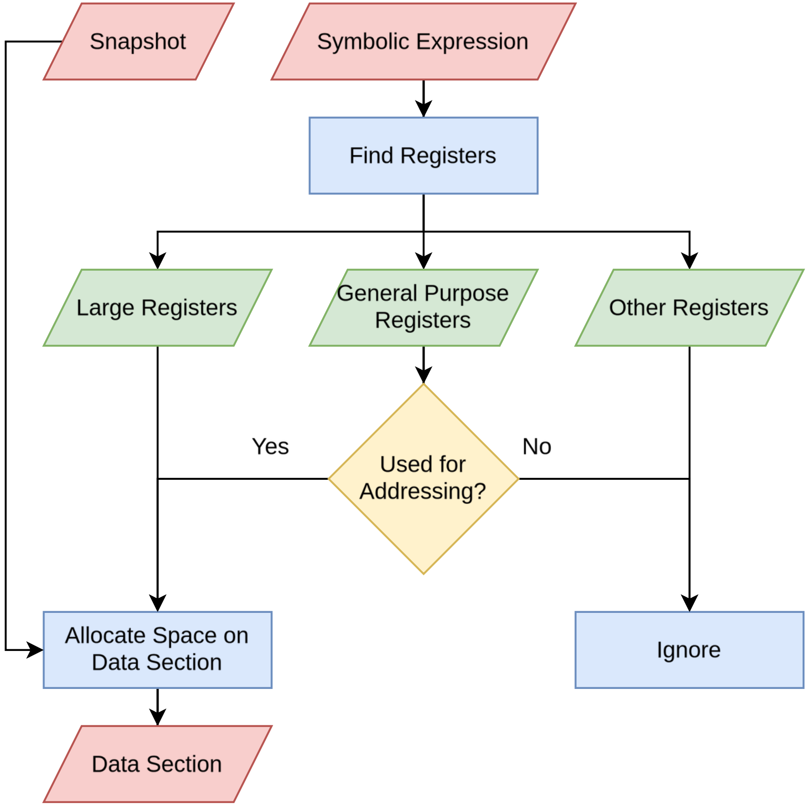
Implementation

- Leverage symbolic expressions to detect the necessary parts

- Resolve addresses in memory read and writes

- Set up:
  - Memory values
  - Registers
  - Stack

- Produce a minimal program



Outline

- ~~Motivation~~
- ~~Background~~
- ~~Design~~
- ~~Implementation~~
- Evaluation
  - Experiment Setup
  - Evaluation
  - Summary

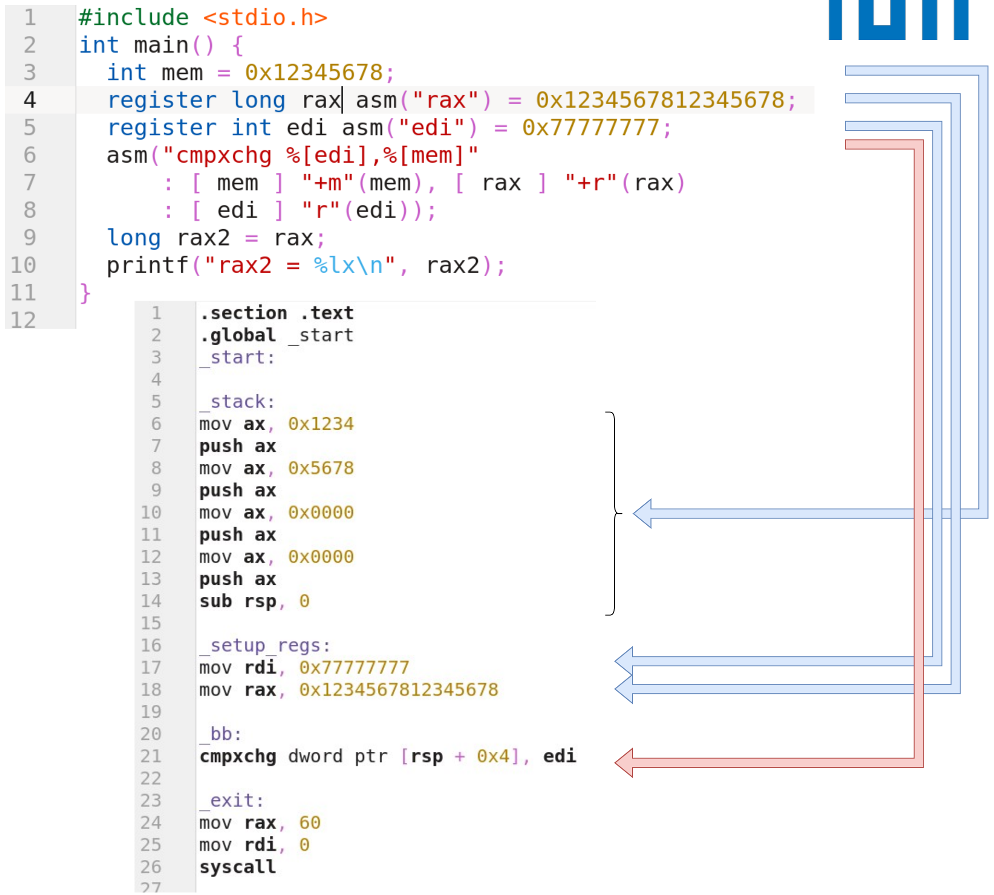
Experiment Setup

- Try to reproduce bugs that are in QEMU
  - Compile older QEMU version
  - Trigger the bug
  - Create a reproducer binary
  - Did it trigger the same bug?

Evaluation: Good Case

- Bug triggered
- Binary size: one-sixth (no clib)
- Symbolic trace: one 57th
- Emulator log: one 122th

- Result:
  - ✔ Bug isolated
  - ✔ Easier to analyze binary
  - ✔ Test that can be reused



```c
#include <stdio.h>
int main() {
  int mem = 0x12345678;
  register long rax asm("rax") = 0x1234567812345678;
  register int edi asm("edi") = 0x77777777;
  asm("cmpxchg %[edi],%[mem]"
      : [ mem ] "+m"(mem), [ rax ] "+r"(rax)
      : [ edi ] "r"(edi));
  long rax2 = rax;
  printf("rax2 = %lx\n", rax2);
}
```

```asm
.section .text
.global _start
_start:

_stack:
mov ax, 0x1234
push ax
mov ax, 0x5678
push ax
mov ax, 0x0000
push ax
mov ax, 0x0000
push ax
sub rsp, 0

_setup_regs:
mov rdi, 0x77777777
mov rax, 0x1234567812345678

_bb:
cmpxchg dword ptr [rsp + 0x4], edi

_exit:
mov rax, 60
mov rdi, 0
syscall
```

cmpxchg should not touch the accumulator in case the values are equal

Evaluation: Bad Case 1

- Output of reproducer is wrong!
- Why?
  - Reproducer depends on verifier
  - Verifier depends on Miasm
  - Not all instructions are implemented

```c
void main() {
    asm("blsi rax, rbx");
}
```

CF is set if the source is not zero

- Result:
  - ✗ Bug cannot be reproduced
  - ✗ Missing instructions need to be implemented

Evaluation: Bad Case 2

- No bugs triggered, program exited succesfully
- Why?
  - Register, stack and memory values are correct?
  - Environment setup is not good enough!

- Result:
  - ✗ A stricter environment is needed
  - ✗ Probably more special cases

Summary & Future Work

- Isolating bugs from emulators is feasible
- A symbolic execution engine that implements all instructions is needed
- There are bugs that require stricter environments, this requires improvements on the reproducer, but no silver bullet

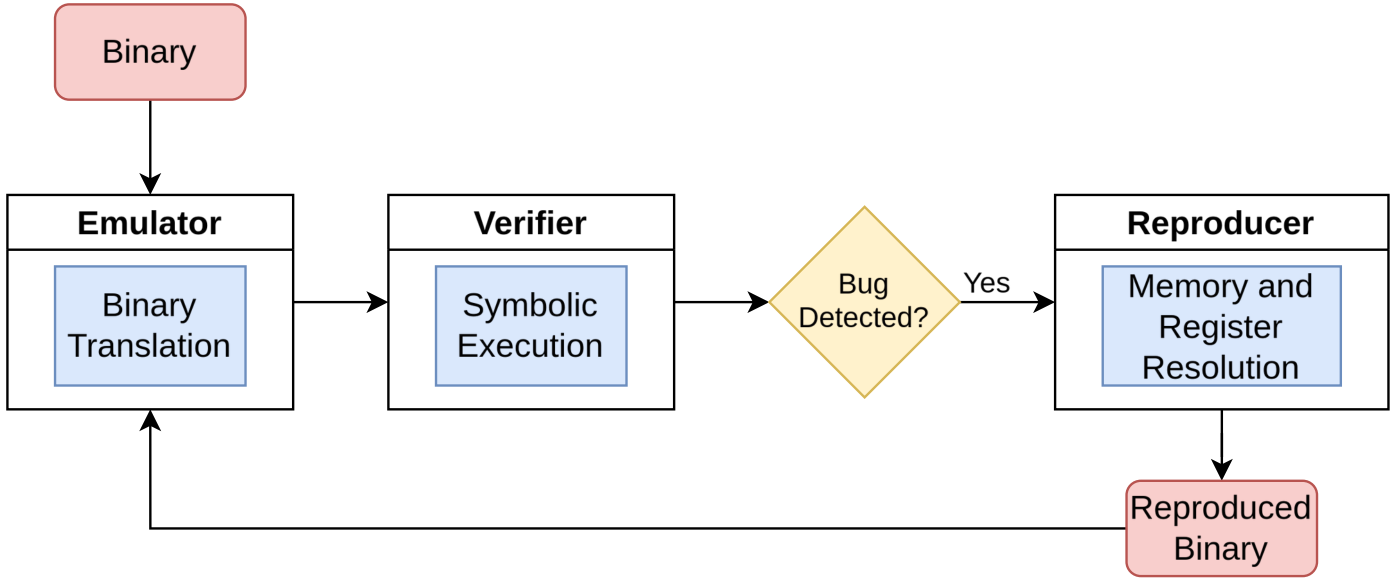
Try it out!

https://github.com/TUM-DSE/focaccia/

Backup

Problem statement

How to automatically isolate bugs that occur during translation


Binary
Emulator
Binary Translation
Verifier
Symbolic Execution
Bug Detected?
Yes
Reproducer
Memory and Register Resolution
Reproduced Binary