Generating Test Suites for GPU Instruction Sets through Mutation and Equivalence Checking

Shoham Shitrit and Sreepathi Pai
University of Rochester

FUZZING 2022

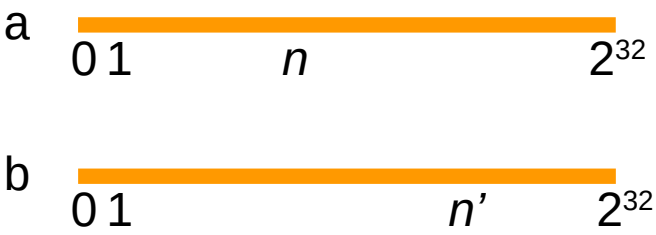
The PTX Instruction Set

- Full-fledged virtual ISA for NVIDIA GPUs
  - Arithmetic, Logic, Control, Memory, Vector, Synchronization, etc.
  - Needs to be compiled to hardware ISA (“SASS”)
- NVIDIA provides an informal specification
- We’re working on a formal semantics for PTX instructions (nearly done!)
- How do we check our semantics?

Method #1: Test against hardware

- Generate C and CUDA tests for each PTX instruction
  - C implements our semantics
  - CUDA runs on the GPU
- Generate test inputs based on types of arguments
  - Use stratified sampling
- Outputs of C and CUDA versions must be equal

```
c = add.u32(a, b)
```



$a \times b = \{(0,0),\ldots,(1,0),\ldots,(n,n'),\ldots,(2^{32},2^{32})\}$

Method #2: Translation Validation

- PTX is compiled to SASS
  - Both must be equivalent
- Does the SASS program implement the semantics of the PTX instruction?
  - Uses our separately developed SASS ISA semantics
- Checked by encoding PTX and SASS programs into SMT formulae

PTX

```
add.u16 %rs1,%rs2,%rs3;
```

SASS

```
LDC.U16 R0, ...
LDC.U16 R4, ...
...
IADD R4, R0, R4
ST.E.U16 [...], R4
```

Testing vs Translation Validation

- Testing and translation validation caught many bugs in our semantics
- Also detected many CUDA compiler bugs!
- But translation validation caught a bug that slipped past testing

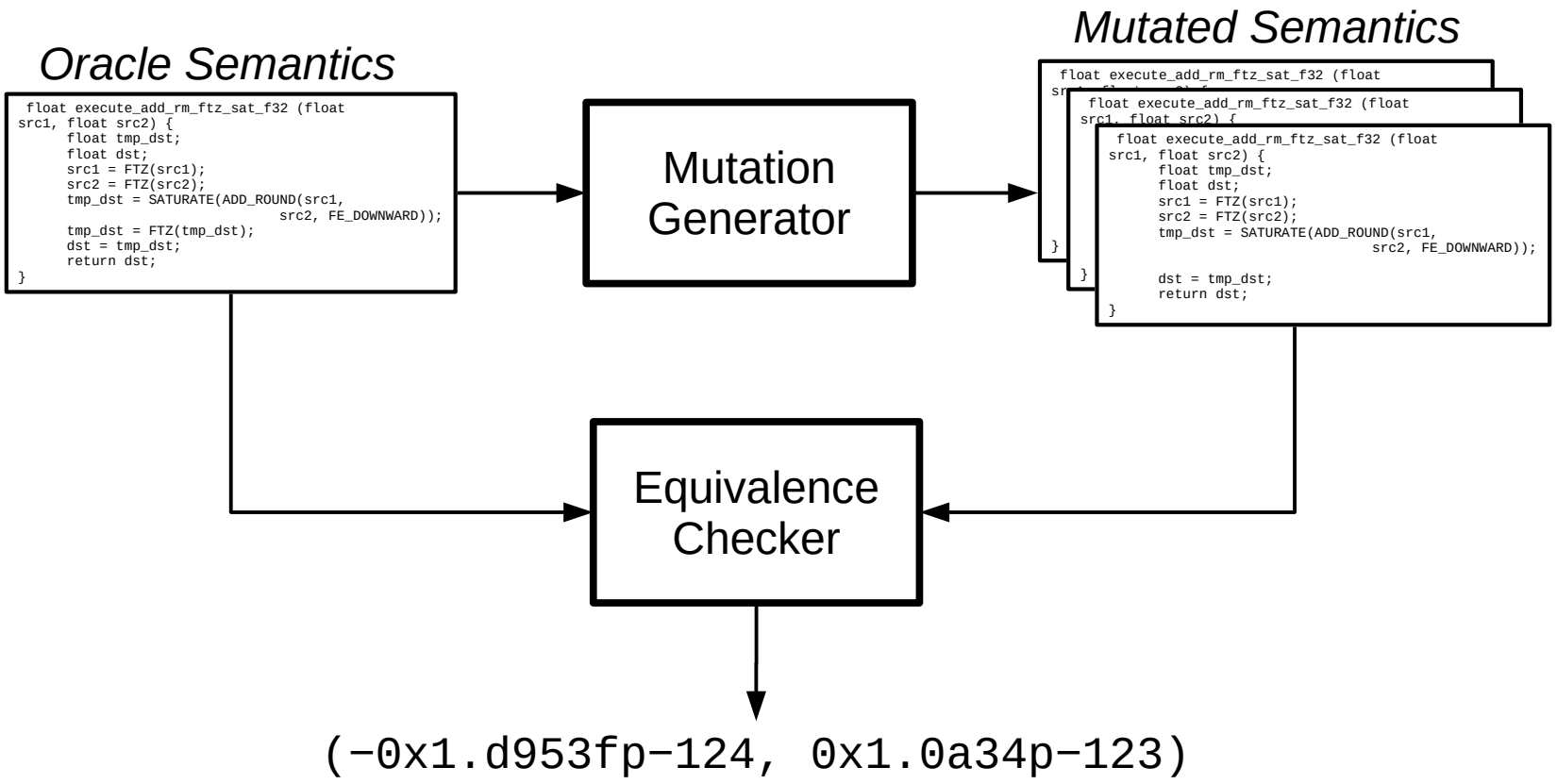
```
 float execute_add_rm_ftz_sat_f32 (
float src1, float src2) {
    float tmp_dst;
    float dst;
    src1 = FTZ(src1);
    src2 = FTZ(src2);
    tmp_dst = SATURATE(ADD_ROUND(src1,
                        src2, FE_DOWNWARD));
    tmp_dst = FTZ(tmp_dst);
    dst = tmp_dst;
    return dst;
}
```

Only detected when inputs are normal floats that sum to a +ve subnormal float

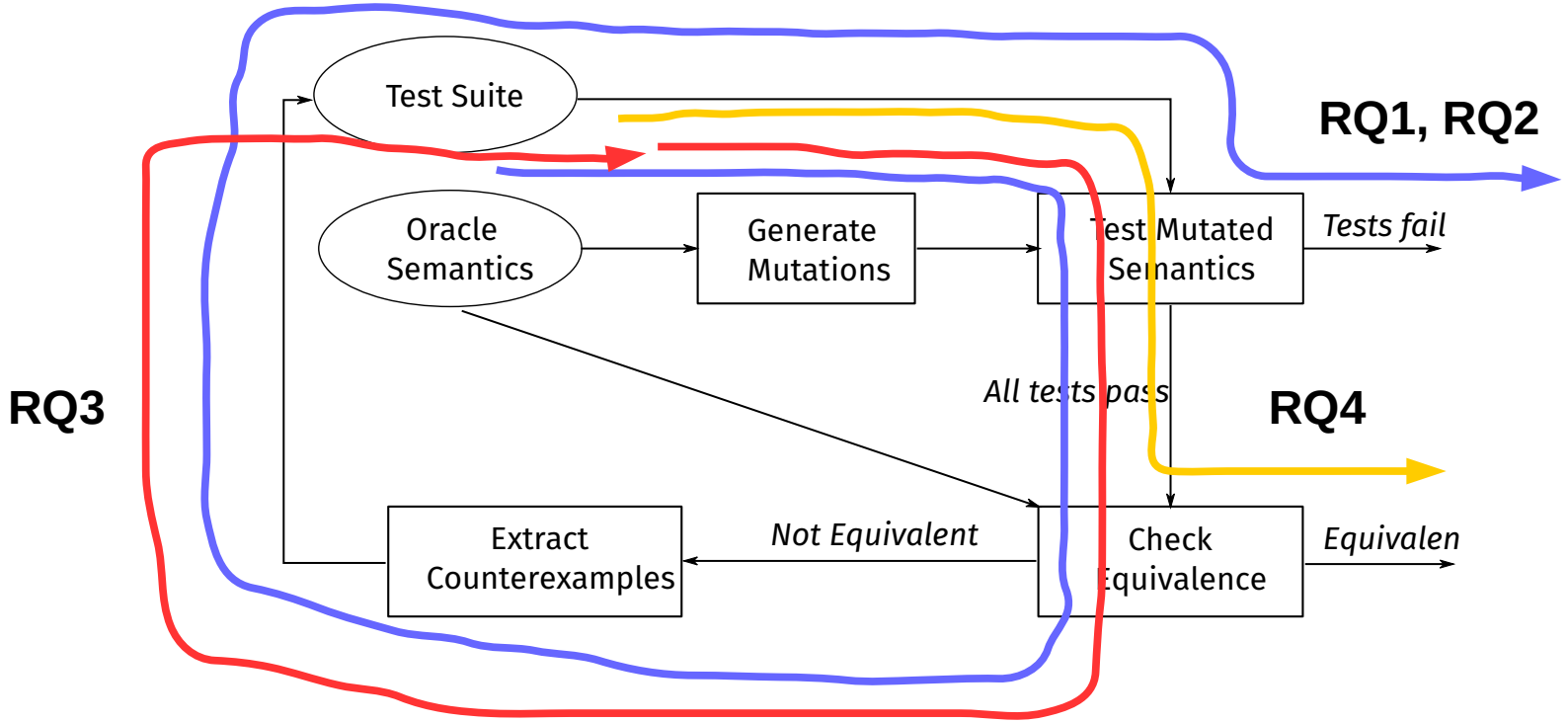
Automated Test Generation using Instruction Set Semantics

- Can we use the correct, formal semantics for generating test inputs?
- Would allow validating implementations that are hard to verify:
  - Compilers
  - Architectural Simulators
  - Hardware GPUs

Basic Test Generation Pipeline



Full Pipeline



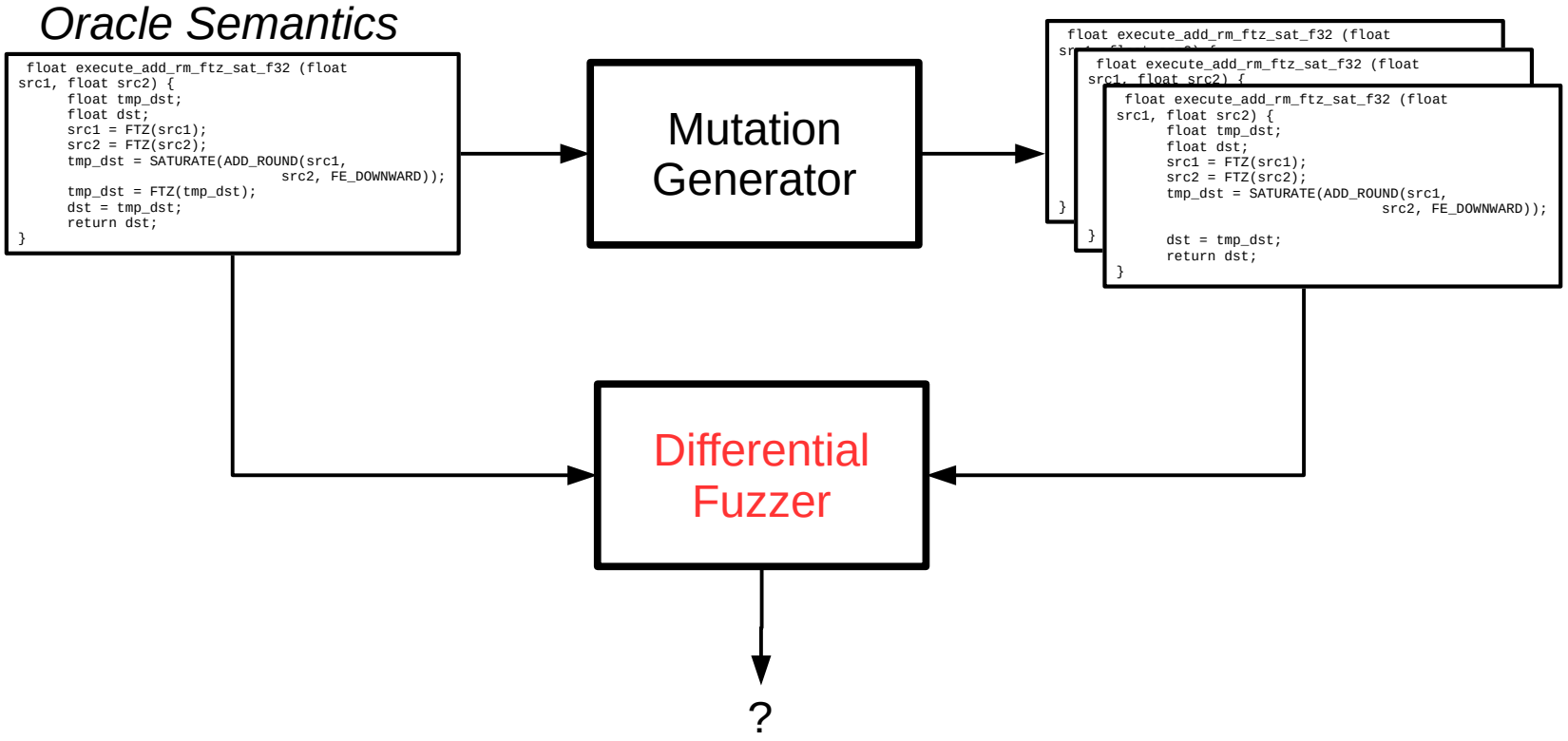
Experimental Setup

- Approximately 4K PTX ALU Instruction Semantics
  - Only 12 used in preliminary experiments
- Mutations using MUSIC
  - Source-to-source mutation engine
- Equivalence Checking using CBMC
  - Bounded model checker for C programs with good floating point support

Preliminary Results

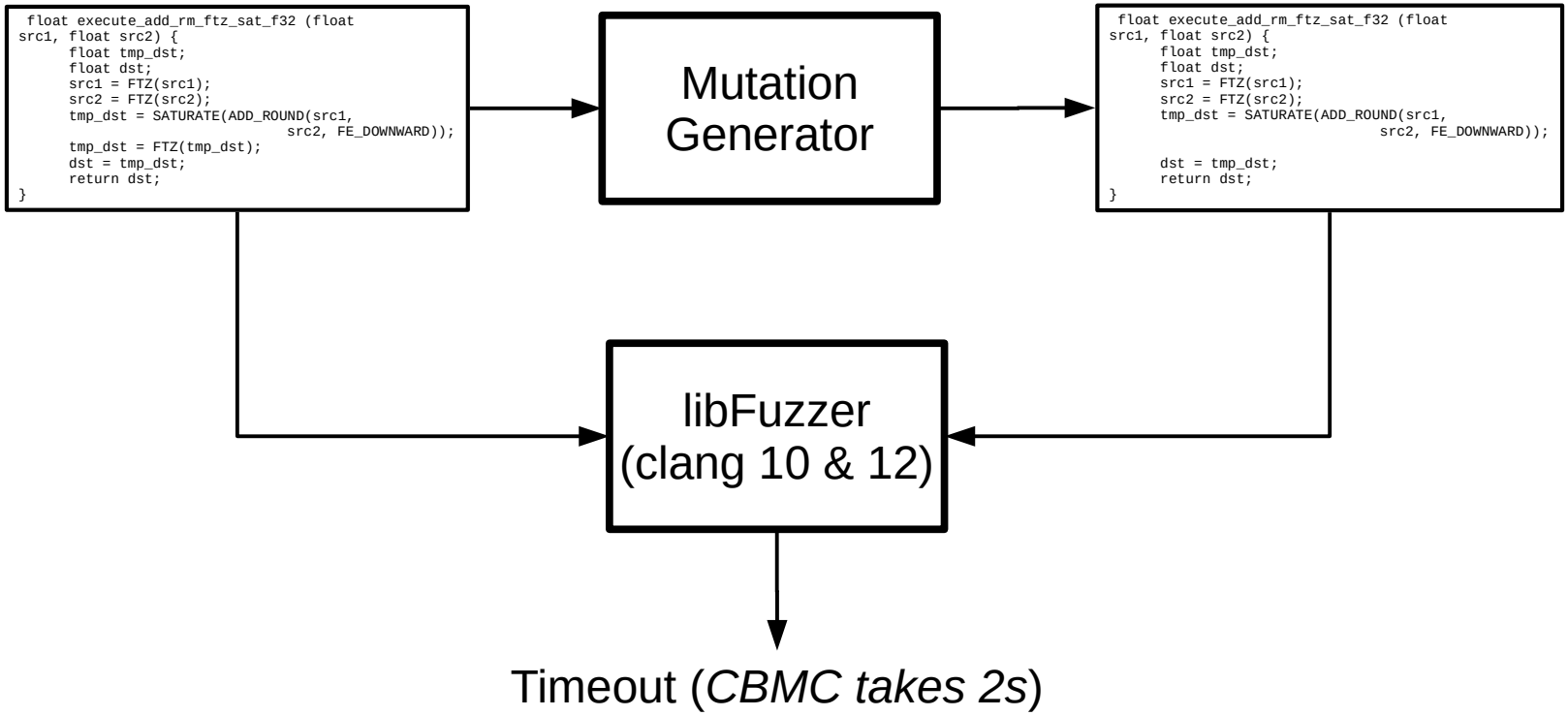
- RQ1, RQ2: Basic pipeline works, can generate buggy version and inputs
- RQ3: Some mutants are “not equivalent”, but testing cannot catch them
- RQ4: Some mutants are syntactically different, but semantically equivalent
- RQ5 (Cost): Equivalence checking is the most expensive part of the pipeline (1 equivalence check every few seconds)
- RQ8 (Test Generation): Augmenting existing test suite is cheaper than generating one from scratch.
- RQ6 (Applicability): No results yet, does equivalence checking work for all instructions?
- RQ7 (Alternatives): No results yet, evaluates tools other than MUSIC/CBMC

RQ9: Differential Fuzzing



RQ9: Very Preliminary Results

Oracle Semantics



Thank you!
Questions?